TURING 图灵计算机科学丛书

PEARSON

计算机科学概论 第11版

Computer Science

[美] J. Glenn Brookshear 著



人民邮电出版社
POSTS & TELECOM PRESS

Computer Science An Overview Eleventh Edition

计算机科学概论 第11版

本书多年来一直深受世界各国高校师生的欢迎，是美国哈佛大学、麻省理工学院、普林斯顿大学、加州大学伯克利分校等著名大学的首选教材，对我国的高校教学也产生了广泛影响。

本书以历史眼光，从发展的角度、当前的水平以及现阶段的研究方向等几个方面，全景式描绘了计算机科学各个子学科的主要领域。在内容编排上，很好地兼顾了学科广度和主题深度，把握了最新的技术趋势。本书用算法、数据抽象等核心思想贯穿各个主题，并且充分展现了各主题的历史背景、发展历程和新的技术趋势，培养读者的大局观，为其今后深入学习其他计算机专业课程打下坚实的基础。

本书深入浅出、图文并茂，内容引人入胜，极易引发读者的兴趣，绝无一般教材的枯燥和晦涩。此外，本书的教学手段多样、习题丰富，并且每章后都附有与本章内容相关的社会现实问题供读者思考和讨论，这些都很好地体现了作者强调培养学生分析问题能力的教学理念。

第11版新增了手持移动设备，特别是智能手机的相关内容，主要涉及第3章（操作系统）、第4章（组网）、第6章（编程语言）和第7章（软件工程）。此外，书中还对软件所有权和责任、训练人工神经网络等内容做了更新，以反映最新技术面貌。

J. Glenn Brookshear 世界知名的计算机科学教育家。他在1975年获得新墨西哥州立大学博士后，创办了Marquette大学的计算机科学学位项目，并在该校任教至今。他的主要研究方向是计算理论。除了本书之外，他还著有*Theory of Computation: Formal Languages, Automata, and Complexity*。

- 计算机科学的全景式展现
- 首屈一指的导论性教材
- 经典传承，新知荟萃



PEARSON

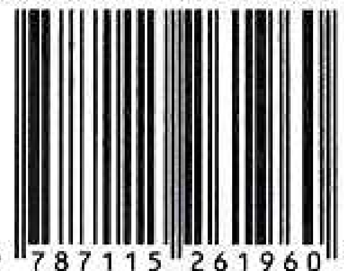
www.pearsonhighered.com

图灵社区：www.ituring.com.cn
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机/计算机科学

人民邮电出版社网址：www.ptpress.com.cn





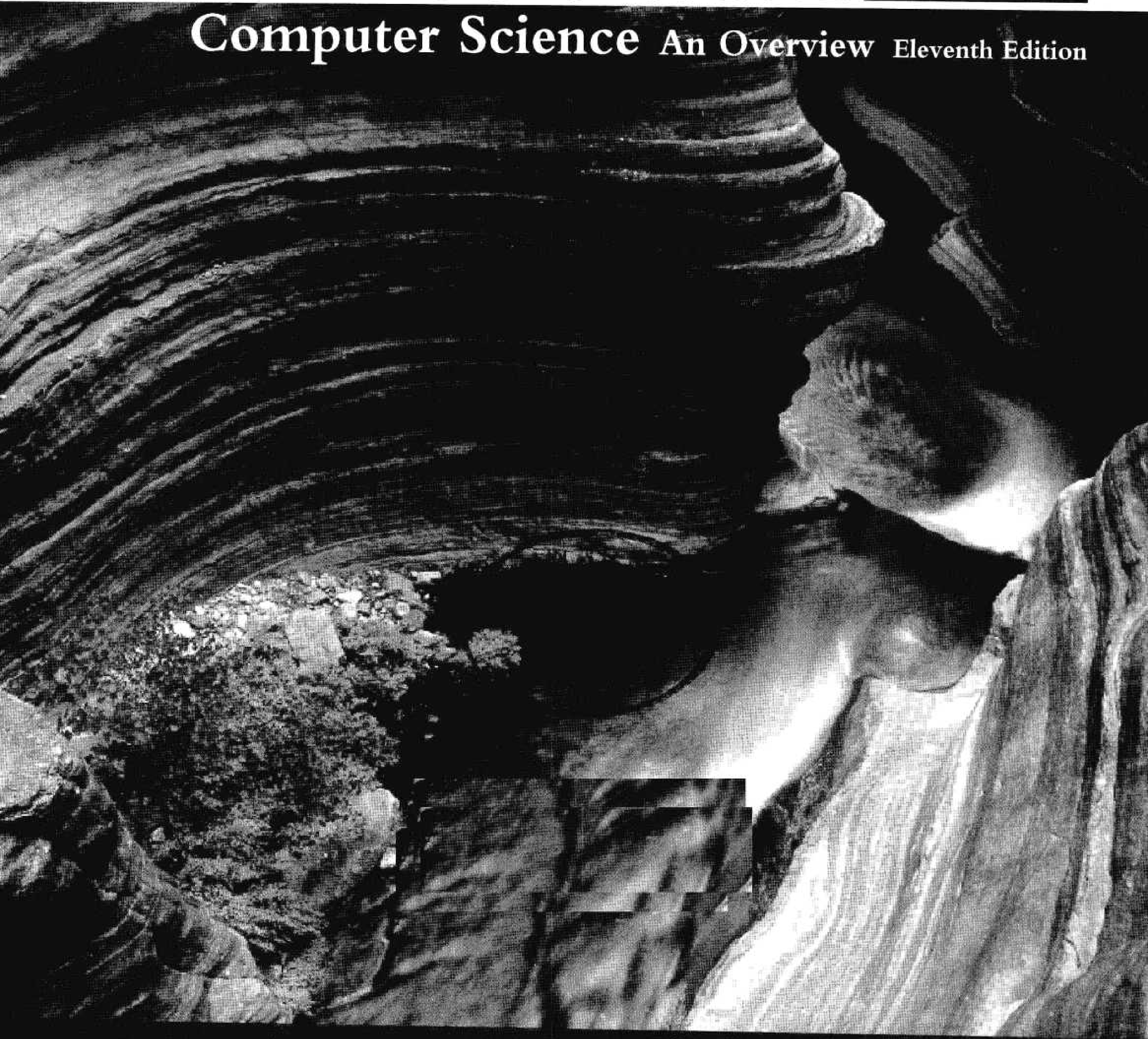
ISBN 978-7-115-26196-0

定价：69.00元

TURING 图灵计算机科学丛书

计算机科学概论 第11版

Computer Science An Overview Eleventh Edition



[美] J. Glenn Brookshear 著

刘艺 肖成海 马小会 毛倩倩 译

图书在版编目（C I P）数据

计算机科学概论 ：第11版 / （美）布鲁克希尔
(Brookshear,J.G.) 著 ；刘艺等译. -- 北京 ：人民邮
电出版社，2011.10
（图灵计算机科学丛书）
书名原文：Computer Science : An Overview ,
Eleventh Edition
ISBN 978-7-115-26196-0

Ⅰ. ①计… Ⅱ. ①布… ②刘… Ⅲ. ①计算机科学
Ⅳ. ①TP3

中国版本图书馆CIP数据核字(2011)第174089号

内 容 提 要

本书是计算机科学概论课程的经典教材，全书对计算机科学做了百科全书式的精彩阐述，充分展现了计算机科学的历史背景、发展历程和新的技术趋势。本书首先介绍的是信息编码及计算机体系结构的基本原理（第 1 章和第 2 章），进而讲述操作系统（第 3 章）和组网及因特网（第 4 章），接着探讨了算法、程序设计语言及软件工程（第 5 章至第 7 章），然后讨论数据抽象和数据库（第 8 章和第 9 章）方面的问题，第 10 章通过图形学讲述计算机技术的一些主要应用，第 11 章涉及人工智能，第 12 章通过对计算理论的介绍来结束全书。本书在内容编排上由具体到抽象逐步推进，很适合教学安排，每一个主题自然而然地引导出下一个主题。此外，书中还包含大量的图、表和示例，有助于读者对知识的了解与把握。

本书适合用作高等院校计算机以及相关专业本科生的教材。

图灵计算机科学丛书

计算机科学概论（第11版）

◆ 著 [美] J. Glenn Brookshear
译 刘 艺 肖成海 马小会 毛倩倩
责任编辑 卢秀丽
执行编辑 毛倩倩 李 静

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1092 1/16
印张：26.5
字数：786千字 2011年10月第1版
印数：1－5 000册 2011年10月河北第1次印刷

著作权合同登记号 图字：01-2011-2960号

ISBN 978-7-115-26196-0

定价：69.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223

反盗版热线：(010)67171154

版 权 声 明

前　言

本书是计算机科学的入门教材。在力求保持学科广度的同时，还兼顾深度，以对所涉及的主题给出中肯的评价。

读者对象

本书面向计算机科学以及其他各个学科的学生。大多数计算机科学专业的学生在最初的学习中都有这样一个误解，认为计算机科学就是程序设计、网页浏览以及因特网文件共享，因为这基本上就是他们所看到的一切。实际上计算机科学远非如此。因此，在入门阶段，学生们需要了解他们主攻的这门学科所涉及内容的广度，这也正是本书的宗旨。本书力图使学生对计算机科学有一个总体的了解，希望在这个基础上，他们可以领会该领域今后其他课程的特点以及相互关系。事实上，本书采用的综述方式也是自然科学入门教程的常用模式。

其他学科的学生如果想融入这个技术化社会，也需要具备这些宽泛的知识背景。适用于他们的计算机科学课程提供的应该是对整个领域很实用的剖析，而不仅仅是培训学生如何上网和使用一些流行的软件。当然这种培训也有其适用的地方，但本书的目的不在于此，而是用作计算机科学的教科书。

因此，在写这本书时，我们力求保持其对非技术类学生的可读性。因此，先前的版本已经被很成功地用作教科书，读者囊括了从高中生到研究生的各个教育层次众多专业的学生。这一版仍将贯彻这一目标。

第 11 版新增的内容

第11版主要增加关于手持移动设备的内容，特别是智能手机。因此，这一版的内容经过了修改、扩充，目的便是呈现所述主题与智能手机技术之间的关系。具体主题包括：

- 智能手机硬件；
- 3G网络与4G网络的区别；
- 智能手机操作系统；
- 智能手机软件开发；
- 智能手机的人机交互界面。

以上新增内容主要出现在第3章（操作系统）和第4章（组网），第6章（编程语言）和第7章（软件工程）中也有提及。

这一版中的其他明显变动包括对以下内容的更新。

- 软件所有权和责任：这一版重写并更新了第7章（软件工程）中与此主题相关的内容。
- 训练人工神经网络：相关内容（见第11章，人工智能）已经实现了现代化。

最后，这一版更新了全书内容来反应当今的技术状况。这主要体现在第0章（绪论）、第1

章（数据存储）和第2章（数据操作）。

章节安排

本书主题由具体到抽象逐步推进——这是一种很利于教学的顺序，每一个主题自然而然地引导出下一个主题。首先介绍的是信息编码、数据存储及计算机体系结构的基本原理（第1章和第2章），进而是操作系统（第3章）和计算机网络（第4章），接着探讨了算法、程序设计语言及软件开发（第5章至第7章），然后探索如何更好地访问信息（第8章和第9章），第10章讲述计算机图形学技术的一些重要应用，第11章涉及人工智能，第12章通过对计算理论的介绍来结束全书。

本书编排顺序自然连贯，但各个章节具有很强的独立性，可以单独查阅，也可以根据不同学习顺序重新排列。事实上，本书通常作为各类课程的教材，内容选择的顺序是多种多样的。其中一种教法是先介绍第5章和第6章（算法和程序设计语言），然后按照需要返回到前面相应章节。我还知道有人是从第12章有关可计算性的内容开始的。这本书还曾作为深入不同领域项目的基础，用于"高级研讨班"的教科书。对于不需要了解太多技术的学生，教学中可以重点讲述第4章（组网及因特网）、第9章（数据库系统）、第10章（计算机图形学）和第11章（人工智能）。

每章开篇都用星号标出了选学章节。选学章节要么是讨论更专业的话题，要么是对传统内容作深入探究。此举仅仅是为那些想采取不同阅读顺序的人提供一点建议。当然，还有其他读法。尤其对于那些寻求快速阅读的读者，我建议采取下面的阅读顺序。

章　节	主　题
1.1～1.4	数据编码和存储基础
2.1～2.3	计算机体系结构和机器语言
3.1～3.3	操作系统
4.1～4.3	组网及因特网
5.1～5.4	算法和算法设计
6.1～6.4	程序设计语言
7.1～7.2	软件工程
8.1～8.3	数据抽象
9.1～9.2	数据库系统
10.1～10.2	计算机图形学
11.1～11.3	人工智能
12.1～12.2	计算理论

在本书中有几条贯穿始终的主线。主线之一是计算机科学是不断发展变化的。本书从历史发展的角度反复呈现各个主题，讨论其当前的状况，并指出研究方向。另一条主线是抽象的作用以及用抽象工具控制复杂性的方式。该主线在第0章引入，然后在操作系统、体系结构、组网、算法、程序设计语言、软件工程、数据组织和计算机图形学等内容中反复体现。

致教师

本教材所包含的内容很难在一个学期内讲授完，因此一定要果断地砍掉不适合自己教学目标的那些主题，或者根据需要重新调整讲授顺序。你会发现，尽管本书有它固有的结构体系，但各个主题在很大程度上是相对独立的，完全可以根据需要作出选择。我写本书的目的是把它

作为一种课程的参考书，而非圈定课程的内容。我希望你把某些主题留作阅读作业，鼓励学生自己学习，而不在课堂讲授。如果我们认为所有的东西都一定要在课堂上讲，那就低估学生的能力了。我们应该教会他们独立学习。

关于本书从具体到抽象的组织结构，我觉得有必要多言几句。作为学者，我们总以为学生会欣赏我们对于学科的观点，这些观点通常是我们在某一领域多年工作中形成的。但作为老师，我认为最好从学生的视角呈现教材。这就是为什么本书首先介绍数据的表示/存储、计算机体系结构、操作系统以及组网，因为这些都是学生们最容易产生共鸣的主题——他们很可能听说过JPEG、MP3这些术语，可能用CD和DVD刻录过资料，买过计算机配件，应用过某一操作系统，或者上过因特网。我发现，从这些主题开始讲授这门课程，学生可以为许多困惑他们多年的问题找到答案，并且把这门课看做是实践课程而不是纯理论的课程。由此出发就会很自然地过渡到较抽象的内容上，例如算法、算法结构、程序设计语言、软件开发方法、可计算性以及复杂性等，而这些内容就是我们本领域的人所认为的计算机科学的主要内容。正如我前面所说的，我并不是强求大家都按此顺序讲课，只是鼓励你们如此尝试一下。

我们都知道，学生能学到的东西要远远多于我们直接传授的内容，而且潜移默化传授的知识更容易被吸收。当要“传授”问题的解决方法时，就更是如此。学生不可能通过学习问题求解的方法变成问题的解决者，他们只有通过解决问题——还不仅仅是那些精心设计过的“教科书式的问题”，才能成为问题的解决者。因此我在本书中加入了大量的问题，并特意让其中一些问题模棱两可——意味着正确方法或正确答案可能不只一个。我建议你们采用并充分拓展这些问题。

另一类适合“潜移默化学习”的主题还有职业精神、伦理和社会责任感。我认为这种内容不应该独立成章，而是应该在有所涉及时讨论，而这正是本书的编排方法。你们会发现，3.5节、4.5节、7.8节、9.7节和11.7节分别在操作系统、组网、软件工程、数据库系统和人工智能的上下文中提及了安全、隐私、责任和社会意识的问题。此外，0.6节就通过总结一些比较著名的理论而引入这一主题——这些理论都企图把伦理上的决断建立在哲学的坚实基础上。你还会发现，每一章都包含了“社会问题”小节，这些问题将鼓励学生思考现实社会与教材内容的关系。

感谢你对本书感兴趣。无论你是否选用本书作为教材，我都希望你认同它是一部好的计算机科学教育文献。

教学特色

本书是多年教学经验的结晶，因此在辅助教学方面考虑较多。最主要的是提供了丰富的问题以加强学生的参与——这一版包含1000多个问题，分为“问题与练习”、“复习题”和“社会问题”。“问题与练习”列在每节末尾（除了第0章外），用于复习刚刚讨论过的内容、扩充以前讨论过的知识，或者提示以后会涉及的有关主题。这些问题的答案可以从图灵社区（www.ituring.com.cn）本书网页免费注册下载。

“复习题”列在每章的末尾（第0章除外）。它们是课后作业，内容覆盖整章，在书中不给出答案。

“社会问题”也列在每章的末尾，供思考讨论。许多问题可以用来开展课外研究，可要求学生提交简短的书面或口头报告。

在每章的末尾还设有“课外阅读”，它列出了与本章主题有关的参考资料。同时，前言以及正文中所列的网址也非常适合查找相关资料。

补充材料

本书的许多补充材料可以从配套网站www.pearsonhighered.com/brookshear上找到。以下内容面向所有读者。

- 每章的实践项目帮助加深理解本教材的主题，并可以帮助了解其他相关主题。
- 每章的“自测题”帮助读者复习本书中的内容。
- 介绍Java和C++基本原理的手册，它在教学顺序上与本书是兼容的。

除此之外，教师还可以登录Pearson Education的教师资源中心（www.pearsonhighered.com）网站申请获得下面的教辅资料。

- 包含“复习题”答案[①]的教师指南。
- PowerPoint幻灯片讲稿。
- 测试题库。

你也许还想看一下我的个人网站www.mscs.mu.edu/~glennb，不是很正式（体现了我某一时的灵感和幽默），但你或许能找到些有用的信息。特别值得一提的是，你将在此找到本书的勘误。

致学生

我有一点点偏执（一些朋友说我可远不是一点点），所以写本书时，我很少接受他人的建议，其中许多人认为一些内容对于初学者过于高深。我相信即使学术界把它们归为“高级论题”，但只要与主题相关就是合适的。读者需要的是一本全面介绍计算机科学的教科书，而不是“缩水”的版本——只包括那些简化了的、被认为适合初学者的主题。因此我不回避任何主题，而是力求寻找更好的解释。我力图在一定深度上向读者展示计算机科学最真实的一面。就好比对待菜谱里的那些调味品一样，你可以有选择地略过本书的一些主题，但我全部呈现出来是为了在你想要的时候供你“品尝”，而且我也鼓励你们去尝试。

我还要指出的是，在任何与技术有关的课程中，当前学到的详细知识未必就适合以后的需要。这个领域是发展变化的——这正是使人兴奋的方面。本书将从现实及历史的角度展现本学科的内容。有了这些背景知识，你们就会和技术一起成长。我希望你们现在就开始行动起来，不要局限于课本的内容，而要进行大胆探索。要学会学习。

感谢你们的信任，选择了我的这本书。作为作者，我有责任创作出值得一读的作品。我希望你们觉得我已经尽到了这份责任。

致谢

首先我要感谢那些支持本书（阅读并使用本书前几个版本）的人们，我感到很荣幸。

David T. Smith（宾夕法尼亚州印第安纳大学）和Dennis Brylow（马凯特大学）对这一版的制作给予了很大帮助。David的贡献主要集中在第0章、第1章、第2章、第7章和第11章，Dennis的贡献主要集中在第3章、第4章、第6章和第10章。如果没有他们的工作，这一版将不会出现。我衷心地感谢他们。

① 也可在图灵社区（www.ituring.com.cn）本书网页免费注册下载。——编者注

我在第10版的前言中已经提到过，要特别感谢Ed Angel、John Carpinelli、Chris Fox、Jim Kurose、Gary Nutt、Greg Riccardi和Patrick Henry Winston在第10版中所做的贡献。他们的努力成果在第11版被保留了下来。

其他对这一版和之前版本作出贡献的人包括J. M. Adams、C. M. Allen、D. C. S. Allison、R. Ashmore、B. Auernheimer、P. Bankston、M. Barnard、P. Bender、K. Bowyer、P. W. Brashear、C. M. Brown、H. M. Brown、B. Calloni、M. Clancy、R. T. Close、D. H. Cooley、L. D. Cornell、M. J. Crowley、F. Deek、M. Dickerson、M. J. Duncan、S. Ezekiel、S. Fox、N. E. Gibbs、J. D. Harris、D. Hascom、L. Heath、P. B. Henderson、L. Hunt、M. Hutchenreuther、L. A. Jehn、K. K. Kolberg、K. Korb、G. Krenz、J. Liu、T. J. Long、C. May、J. J. McConnell、W. McCown、S. J. Merrill、K. Messersmith、J. C. Moyer、M. Murphy、J. P. Myers、Jr.，D. S. Noonan、W. W. Oblitey、S. Olariu、G. Rice、N. Rickert、C. Riedesel、J. B. Rogers、G. Saito、W. Savitch、R. Schlafly、J. C. Schlimmer、S. Sells、G. Sheppard、Z. Shen、J. C. Simms、M. C. Slattery、J. Slimick、J. A. Slomka、D. Smith、J. Solderitsch、R. Steigerwald、L. Steinberg、C. A. Struble、C. L. Struble、W. J. Taffe、J. Talburt、P. Tonellato、P. Tromovitch、E. D. Winter、E. Wright、M. Ziegler，还有一位匿名的朋友。我向他们中的每一位致以最真诚的谢意。

如前所述，本书的配套网站上有Java和C++手册来讲述这两种语言的基础知识，与本书的内容相得益彰。它们是由Diane Christie撰写的，在此表示感谢。另外，还要感谢Roger Eastman，他是本书配套网站上每章实践项目的出题人。

我同时要感谢为本项目作出贡献的Addison-Wesley的员工。他们不仅是很好的合作伙伴，而且是很好的朋友。如果你们打算写一本教材，可以考虑交给Addison-Wesley出版。

我还要感谢我的夫人Earlene和我的女儿Cheryl，感谢她们这么多年对我的鼓励。当然Cheryl已经长大，几年以前已经离家开始独自生活。Earlene还陪在我身边。我是一个幸运的人。1998年12月11日的早晨，我突发心脏病，是她及时把我送到了医院，让我逃过了一劫。（对于年轻一代的你们，我有必要解释一下，躲过心脏病的一劫有点像你们又获准延期提交课后作业。）

最后，我要感谢我的父母，本书即是给他们的献礼。我用下面一句赞美的话作为结束，就不说是他们哪一位说的了：“我们儿子的书真的非常好，人人都应该阅读。”

J. G. B.

目　录

第0章 绪论

在开篇的这一章，我们探讨计算机科学所涉及的领域，介绍其历史背景，然后为我们的深入学习奠定基础。

本章内容

计算机科学这门学科，是要为计算机设计、计算机程序设计、信息处理、问题的算法解决方案和算法过程本身等主题建立科学的基础。计算机科学既是当今计算机应用的支柱，又是今后计算基础设施的基础。

本书将详细介绍计算机科学，探索广阔的主题，包括构成大学计算机科学课程的大部分主题。我们要领略这个领域的博大精深和变化发展。因此，除了这些主题本身，我们还关注它们的历史发展、现今的研究动态以及今后的前景。我们的目标是让人们以学以致用的态度来对待计算机科学——既帮助那些要在此领域继续深入学习的人，也促成其他领域的人在技术不断进步的社会崭露头角。

0.1 算法的作用

首先让我们了解一下计算机科学最基础的概念——“算法”。一般来讲，**算法**（algorithm）是完成一项任务所遵循的一系列步骤。（在第5章，我们将给出比较精确的定义。）例如，有关于烹饪的算法（称为菜谱），有在陌生城市准确定位的算法（通常称为道路指南），有使用洗衣机的算法（通常标示在洗衣机的内盖上或者贴在自助洗衣店的墙上），有演奏音乐的算法（以乐谱的形式表示），还有魔术表演的算法（见图0-1）。

在一台机器（如计算机）执行一项任务之前，必须先找到完成这项任务的算法，并且用与该机器兼容的形式表示出来。某一个算法的表示称作一个**程序**（program）。为了人们读写方便，计算机程序通常打印在纸上或者显示在计算机屏幕上。为了便于机器识别，程序需要采取一种与该机器技术兼容的形式进行编码。开发程序、采取与机器兼容的形式进行编码并将其输入到机器中的过程，称作**程序设计**（programming）。程序及其所表示的算法总称为**软件**（software），而机器设备本身则称为**硬件**（hardware）。

效果：表演者从一副普通的扑克牌中抽取若干张牌，充分洗牌后将牌正面朝下展开在桌面上。然后，表演者会根据观众的要求相应地翻出红牌或者黑牌。

秘诀：

步骤 1 从一副普通扑克牌中抽取10张红牌和10张黑牌。把它们根据颜色分为两摞，正面朝上放在桌面上。

步骤 2 告诉观众你已经选取了若干张红牌和黑牌。

步骤 3 拿起红牌，装作整理成一摞的样子，用左手正面朝下拿好牌，同时用右手的拇指和食指挤压这摞牌的两端，把牌面向下推，使得每张牌呈现向下的弧形。然后，把这摞红牌扣在桌子上并宣布："这是其中的红牌。"

步骤 4 拿起黑牌，模仿步骤3的方法，但要使这些牌呈现向上的弧形。然后，把牌扣在桌子上并宣布："这是其中的黑牌。"

步骤 5 把黑牌放回桌面后，立即用双手把红牌和黑牌混在一起（仍然正面朝下），平铺在桌面上。告诉大家你已经洗好了牌。

步骤 6 只要桌面上还有扣着的牌，可以重复下面的步骤：

6.1 请观众要一张红牌或黑牌；

6.2 如果所要的牌为红色，而且桌面上倒扣有凹形的牌，就翻开其中的一张并告诉大家"这是一张红牌"；

6.3 如果所要的牌为黑色，而且桌面上倒扣有凸形的牌，就翻开其中的一张并告诉大家"这是一张黑牌"；

6.4 否则，就告诉大家桌面上没有所要求颜色的牌了，然后翻开桌面上所有的牌，以证实你的断言。

图0-1 一个魔术的算法

算法的研究起源于数学学科。事实也的确如此，它是数学家的重要活动，远远早于当今计算机的出现。它的目标是找出一组指令，描述如何解决某一特定类型的所有问题。求解两个多位数商的长除算法是早期研究中一个最著名的例子。另一个例子是古希腊数学家欧几里得发现的欧几里得算法——求两个正整数的最大公约数（见图0-2）。

描述：本算法假定输入是两个正整数，目的是要计算这两个数的最大公约数。

过程：

步骤 1 将这两个数中较大的一个和较小的一个分别赋予M和N。

步骤 2 用M除以N，余数设为R。

步骤 3 如果R不为0，那么将N的值赋予M，并将R的值赋予N，然后回到步骤2；否则最大公约数就是N当前被赋予的值。

图0-2 求两个正整数的最大公约数的欧几里得算法

一旦我们找到了执行一个任务的算法，那么在执行该任务时，就不再需要了解该算法所依据的原理——任务的完成演变成了遵照指令操作的过程。（不需要了解算法的工作原理，我们就可以根据长除算法求商，或者根据欧几里得算法求得最大公约数。）在某种意义上，解决这个问题的智能被编码到算法中。

我们能够设计出那些执行有用任务的机器，是因为我们有上述能力通过算法来捕获和传达智能（至少是智能行为）。因此，机器的智能级别受限于算法所传达的智能。只有存在执行某一项任务的算法时，我们才可以制造出执行这一任务的机器，换言之，如果我们找不到一个解决某问题的算法，那么这个问题的解决就超出了机器的能力范围。

20世纪30年代，库尔特·哥德尔（Kurt Gödel）发表了不完备性定理的论文，它使确定算法能力的局限性成为数学的一个研究课题。这个定理的主旨就是，在任何一个包括传统意义的算术系统的数学理论内，总有一些命题的真伪无法通过算法的手段来确定。简言之，对于我们算术系统的任何全面研究都超越了算法活动的能力。

这一认识动摇了数学的基础，于是关于算法能力的研究随之而来，它开创了今天计算机科学这门学科。的确，正是算法的研究构成了计算机科学的核心。

0.2 计算机器的由来

今天的计算机有着庞大久远的世系渊源，其中较早的一个计算设备是算盘。历史告诉我们，它很可能源于中国古代且曾被用于早期希腊和罗马文明。算盘本身非常简单，一个矩形框里固定着一组小棍，而每个小棍上又各串有一组珠子（见图0-3）。在小棍上，珠子上下移动的位置就表示所存储的值。正是这些珠子的位置代表了这台“计算机”所表示和存储的数据。这台机器是依靠人的操作来控制算法执行的。因此，算盘自身只算得上一个数据存储系统，它必须在人的配合下才成为一台完整的计算机器。

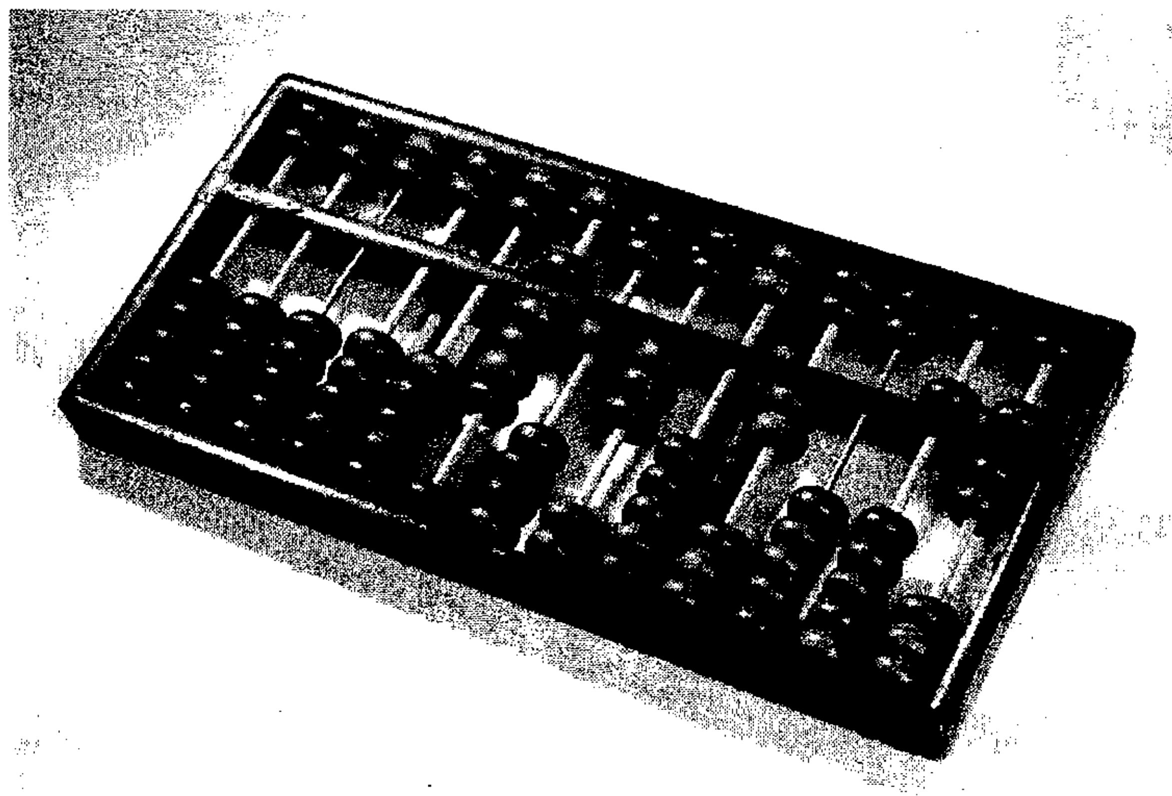


图0-3 算盘（Wayne Chandler拍摄）

从中世纪到近代，人们开始探求更复杂的计算机器。后来，一些发明人开始基于齿轮技术设计计算机器。采用这种技术的发明家有法国的布莱斯·帕斯卡尔（Blaise Pascal，1623—1662）、德国的戈特弗里德·威尔赫尔姆·莱布尼茨（Gottfried Wilhelm Leibniz，1646—1716）和英国的查尔斯·巴贝奇（Charles Babbage，1792—1871）等。这些机器利用齿轮的位置来表示数据，要在规定齿轮初始位置的基础上机械地输入数据。帕斯卡尔和莱布尼茨的机器所计算的结果是通过观察齿轮的最终位置得到的。而巴贝奇构想了这样一种机器，可以把计算的结果打印在纸上，从而消除可能出现的誊写错误。

就执行算法的能力而言，我们可以看到这些机器在灵活性上的进步。帕斯卡尔的机器只是为了执行加法而设计，因此必须将正确的步骤序列嵌入到机器结构本身。同样，莱布尼茨的机器也把算法嵌入在其体系结构中，但它提供了多种算术运算供操作员选择。巴贝奇的差分机仅造了一个演示模型，可以被修改以执行各种计算，但他设计的分析机（因一直没有得到任何基金支持而未生产出来）则能够在纸卡片上读取以洞孔形式表示的指令。所以，巴贝奇的分析机是可编程的。事实上，奥古斯塔·艾达·拜伦（Augusta Ada Byron）通常被称为世界上第一位程序员，她曾发表过一篇论文，阐述巴贝奇的分析机如何编程并实现各种各样的计算。

巴贝奇的差分机

查尔斯·巴贝奇设计的这台机器的确是现代计算机设计的先驱。如果能用比较经济的技术制造出这台机器，如果当时商业和政府数据处理需求达到今天的规模，那么巴贝奇的思想可能在19世纪就引发了计算机革命。事实上，在他有生之年只是造出了差分机的演示模型。该机器通过“逐次差分”的计算来决定数字值。我们来研究一下计算整数平方的问题，帮助我们加深对这一技术的理解。首先我们从基础知识开始，0的平方是0，1的平方是1，2的平方是4，3的平方是9。据此，可以按照下面的方法得到 4 的平方（见下图）。首先我们来计算一下已知平方之间的差：$1^2-0^2=1$，$2^2-1^2=3$，$3^2-2^2=5$。然后，我们计算这些结果的差：$3-1=2$，$5-3=2$。注意看，这些差都是2。假设这个规律能够成立（数学上可以证明它是成立的），那么我们可以得出这样的结论：（4^2-3^2）和（3^2-2^2）之间的差也一定是2。因为（4^2-3^2）比（3^2-2^2）大2，所以$4^2-3^2=7$，$4^2=3^2+7=16$。现在，我们已经知道了4的平方，那么就可以依据1^2、2^2、3^2和4^2的值继续计算5的平方。（虽然更深入地讨论逐次差分已经超出了本书范围，但是学过微积分的学生可能已观察到，前面的例子是基于这样的事实：$y=x^2$的二阶导数是一条直线。）

x	x^2	一阶差分	二阶差分
0	0	1	2
1	1	3	2
2	4	5	2
3	9	7	2
4	16		
5			

奥古斯塔·艾达·拜伦

洛夫莱斯伯爵夫人奥古斯塔·艾达·拜伦是计算界关注的焦点人物。艾达·拜伦的一生近乎悲惨，去世时还不到37岁（1815—1852）。她体弱多病，身处限制妇女从业的社会，还是个新教教徒。尽管对广泛的科学感兴趣，但她还是专注于数学研究。1833年，目睹了查尔斯·巴贝奇的差分机样机演示后，她就被这台机器迷住了。她把一篇讨论巴贝奇分析机设计的论文从法文翻译为英文，算是她最早对计算机科学所做的贡献。巴贝奇还鼓励她在翻译中增加一个附录，介绍该机器的应用，并提供了例子说明该机器如何进行编程以实现各种各样的任务。巴贝奇对艾达·拜伦的工作十分热情，这是因为他希望论文的出版可以帮助他得到资金援助，以建造他的分析机。（作为拜伦勋爵的女儿，艾达·拜伦具有名人的地位，在生意场上也有一定的关系。）巴贝奇最终也没有得到资金援助，但是艾达·拜伦的附录保存了下来。人们认为该附录包含了第一批计算机程序的例子。关于巴贝奇对艾达的工作影响有多大，研究计算机历史的学者们一直争论不休。有些历史学家认为巴贝奇作出了重大贡献，但另外一些人则认为巴贝奇并没有帮到艾达，从很大程度上来看反而是一种阻碍。无论如何，奥古斯塔·艾达·拜伦都被认为是世界上第一位计算机程序员，美国国防部为了纪念这位伟大的女性，用她的名字命名了一种程序设计语言（Ada）。

通过纸卡片上的洞孔来传达算法的思想并不是源于巴贝奇。他是从约瑟夫·雅卡尔（Joseph Jacquard，1752—1834）那里得到这个想法的。约瑟夫·雅卡尔于1801年研制出一种织布机，它在织布过程中所执行的步骤是由纸卡片上洞孔的样式决定的。因此，织布机的算法很容易进行修改，可以得出不同的编织设计。另一个受益雅卡尔思想的人是赫尔曼·霍尔瑞斯（Herman Hollerith，1860—1929），他灵活运用这一观念——用纸卡片上洞孔的样式来表示信息，加速了美国1890年人口普查中的表格处理。（霍尔瑞斯的这项改造促使了IBM的诞生。）这种卡片最终被称作穿孔卡片并广为人知，直到20世纪70年代仍被作为与计算机交互的流行工具。的确，这项技术至今尚存，在美国2000年总统选举的投票工作中我们还可见到它的身影。

在那个年代，即使有资金的支持，技术上也不足以制造帕斯卡尔、莱布尼茨和巴贝奇的复杂的齿轮驱动的机器。但是，随着20世纪初期电子技术的进步，人们克服了这个障碍。这期间，乔治·斯蒂比兹（George Stibitz）的电子机械机器，于1940年在贝尔实验室里建造；马克一号（Mark I），由霍华德·艾肯（Howard Aiken）和IBM公司的一个工程师小组一起在哈佛大学建造（见图0-4）。这些机器大量使用了电子控制的机械式继电器。从这个意义上说，这些机器几乎是刚造出来就过时了，因为其他研究人员已在应用电子管技术建造完全电子化的计算机。第一台这样的机器显然是Atanasoff-Berry机器，1937～1941年由约翰·阿塔纳索夫（John Atanasoff）和他的助手克利福德·贝利（Clifford Berry）建造于艾奥瓦州立学院（现在的艾奥瓦州立大学）。另一台是称为巨人（Colossus）的机器，在汤米·弗劳尔（Tommy Flowers）的指导下建造于英国，该机器在第二次世界大战后期曾被用来破解德国的情报。（实际上，这类机器有十余台，但是由于军方的保密和国家安全问题而未能列入“计算机家谱”。）不久，更为灵活的机器出现了，如ENIAC（Electronic Numerical Integrator And Calculator，电子数字积分器和计算器），它是由约翰·莫奇利（John Mauchly）和普雷斯波·埃克特（J. Presper Eckert）在宾夕法尼亚大学莫尔电子工程学院研制的。

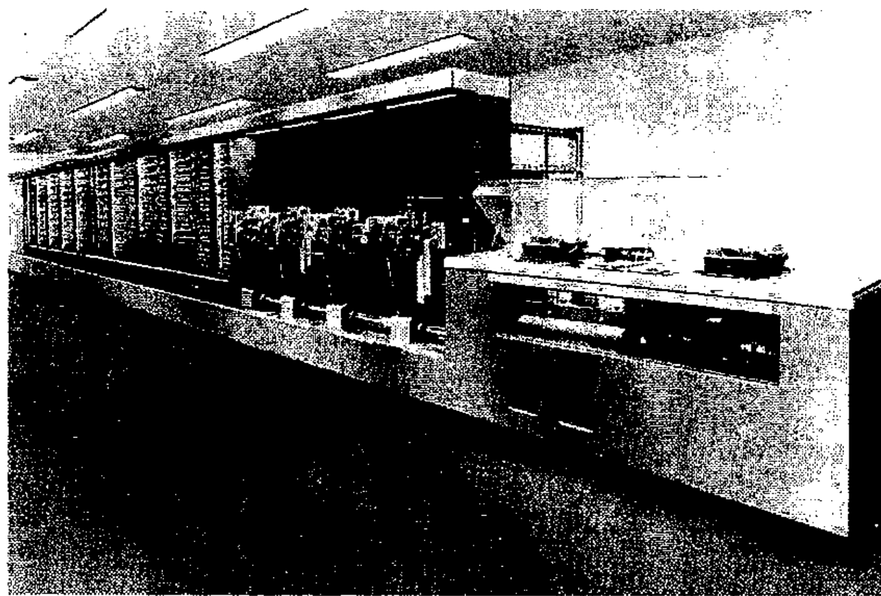


图0-4 马克一号计算机（照片由IBM档案室提供）

从那时起，计算机器的发展史就开始和技术进步紧紧相连，包括晶体管的发明（物理学家William Shockley、John Bardeen和Walter Brattain因此获得了诺贝尔奖）和后来集成电路的开发（Jack Kilby因此荣获了诺贝尔物理学奖）。由于这些技术，以往20世纪40年代房间大小的机器在数十年间缩小到了单机柜大小。与此同时，计算机器的处理能力每两年便会翻倍，而这一趋势一直持续到了今天。随着集成电路技术的进步，计算机中的集成电路被封装在玩具大小的塑料块中做成芯片，可以在电子市场上随处买到。

计算机的普及在很大程度上得益于台式机的发展。这些计算机的起源可以追溯到计算机爱好者通过芯片组合构建家用计算机。正是在这些计算机爱好者的“地下”活动中，史蒂夫·乔布斯（Steve Jobs）和斯蒂芬·沃兹尼亚克（Stephen Wozniak）两个人制造出了有商业价值的家用计算机，并于1976年成立了苹果计算机公司（现称苹果公司）来制造和销售他们的产品。其他经销类似产品的公司有Commodore、Heathkit和Radio Shack等。虽然这些产品在计算机爱好者中很畅销，但是并没有被商业界普遍接受。面对大量的计算需要，这些商家仍然青睐于著名的IBM公司。

1981年，IBM公司推出了它的第一台台式计算机，称为个人计算机或PC，其基础软件由一个称为微软（Microsoft）的年轻公司开发。PC一经推出立即获得了极大的成功，并且奠定了这

种台式计算机在商界人士心目中作为日用品的地位。今天，术语PC已广泛地指称整个这一类机器（来自各个不同厂商），其设计都是从IBM公司最初的台式计算机演变而来，而且它们大多数继续与微软公司的软件一起销售。不过，有时候术语PC也与统称的术语台式机和笔记本电脑互换使用。

在20世纪后期，**因特网**的出现大大改变了人们的沟通方式，这种技术将个人计算机连成了一个全球系统。在这个背景下，Tim Berners-Lee（英国的一位科学家）提出了这样一个系统，它可以通过因特网把计算机上存储的文档链接起来形成错综复杂的链接信息网，这便是**万维网**（World Wide Web），简称Web。为了能够访问Web信息，人们开发了一种叫做**搜索引擎**（search engine）的软件系统，筛选Web上的信息，对结果进行“归类”，然后通过搜索结果帮助用户探求特定内容。这一领域的主要参与者有谷歌、雅虎和微软。这些公司不断扩展其与Web相关的活动，而且经常会挑战我们的传统思维方式。

谷 歌

谷歌创立于1998年，已经成为世界上最受认可的一家技术公司。现在，数百万人使用其核心服务谷歌搜索引擎在万维网上搜索文档。此外，谷歌提供了电子邮件服务（Gmail）和一个基于因特网的视频共享服务（YouTube），以及其他大量因特网服务（包括Google Maps、Google Calendar、Google Earth、Google Books和Google Translate）。

谷歌不仅极具创业精神，而且就不断发展的科技对现代社会的挑战给出了一系列例证。例如，对于谷歌的搜索引擎人们的疑问是，一个国际化公司应该在何种程度上按各独立政府的意愿行事，而对于YouTube则是，一个公司对于他人利用其服务分发的信息应负何种程度的责任。另外，Google Books引起了人们对于知识产权的适用范围和局限性的关注，而Google Maps则被指责侵犯了隐私权。

与此同时，桌面计算机（以及更新的笔记本电脑）正在被人们所接受并用于家用，计算机器的微型化仍在继续。今天，很多设备中都嵌有微型计算机。例如，现在的汽车就有运行GPS（Global Positioning System，全球定位系统）的小型计算机，用于监控引擎的机能，并提供控制音频和电话通信系统的语音命令服务。

也许对于计算机微型化而言，最具革命性的应用在于移动电话的扩展性能。的确，不久之前还只能用于通话的电话都已经演进为小型手持通用计算机，即**智能手机**（smartphone），其中通话仅是众多应用之一。这些“电话”配备有大量传感器和接口，包括照相机、话筒、指南针、触摸屏、加速计（用以检测手机的方向和动作），以及一系列无线技术（以便与其他智能手机和计算机通信）。这一潜力是巨大的。的确，很多人认为智能手机对于社会的影响将大于PC。

计算机的微型化和其功能的日益增多已经把计算机技术推向了当今社会的最前沿。如今，计算机技术非常普及，熟练掌握其应用已经成为现代社会成员的基本要求。计算机技术已经改变了政府施加控制的能力，对全球化经济产生了巨大的影响，导致在科学研究领域出现了一些令人瞩目的成就，革新了数据收集、存储和应用的作用，为人们提供了新的通信和交互方式，不停地挑战社会现状。结果是围绕着计算机科学的学科大量涌现，每门学科现在都成了重要的研究领域。此外，就像很难区分机械工程和物理一样，我们也很难在这些领域与计算机科学之间画出一条分界线。因此，为了获得合适的视角，我们的研究不仅要涉及计算机科学核心的中心主题，而且还将探索与科学应用和影响相关的各种学科领域。因此，对计算机科学的全面介绍必然要涉及其他很多学科的知识。

0.3 算法的科学

数据存储容量有限，程序设计过程复杂而耗时，诸如此类原因限制了早期计算机器所能处理的算法复杂性。如今，随着这些局限性的消除，机器已经被应用到执行越来越艰巨、越来越复杂的任务中。人们企图用算法表达这些任务，但却感到了思维能力上的不足，于是越来越多的工作转向算法和程序设计过程的研究。

在这种情况下，数学家的理论研究开始有了回报。由于哥德尔不完备性定理，数学家已经在研究有关算法过程的问题了，而这正是先进技术目前面临的问题。由此，孕育出了被称作计算机科学的这门学科。

如今，计算机科学已经奠定了它算法科学的地位。这门科学范围很广，涉及数学、工程学、心理学、生物学、商业管理和语言学等多个学科。事实上，研究计算机科学不同分支的研究人员对科学的定义也许会截然不同。例如，计算机体系结构领域中的研究者主要关注微型电路技术，因此他们将计算机科学视为技术的进步和应用；但数据库系统领域的研究者则认为计算机科学就是要寻求方法来提升信息系统的有用性；而人工智能领域的研究者则把计算机科学视为智能和智能行为的研究。

因此，介绍计算机科学必然要包含多个主题，接下来的章节都将始终遵循这种做法。对每一个主题，我们的目标就是要介绍这门学科的核心思想、当前的研究课题以及一些用于本领域中先进知识的技术。在学习过程中，我们很容易因主题太多而忽视整体框架。因此，为了整理思路，我们现在提出一些问题，让大家看到该领域的研究焦点。

- 算法过程可以解决什么样的问题？
- 怎样才能比较容易地找到算法？
- 如何改进表示和传达算法的技术？
- 如何分析和比较不同算法的特征？
- 如何使用算法来操作信息？
- 如何应用算法来产生智能行为？
- 算法的应用对社会有何种影响？

注意，所有这些问题都与算法研究有关（见图0-5）。

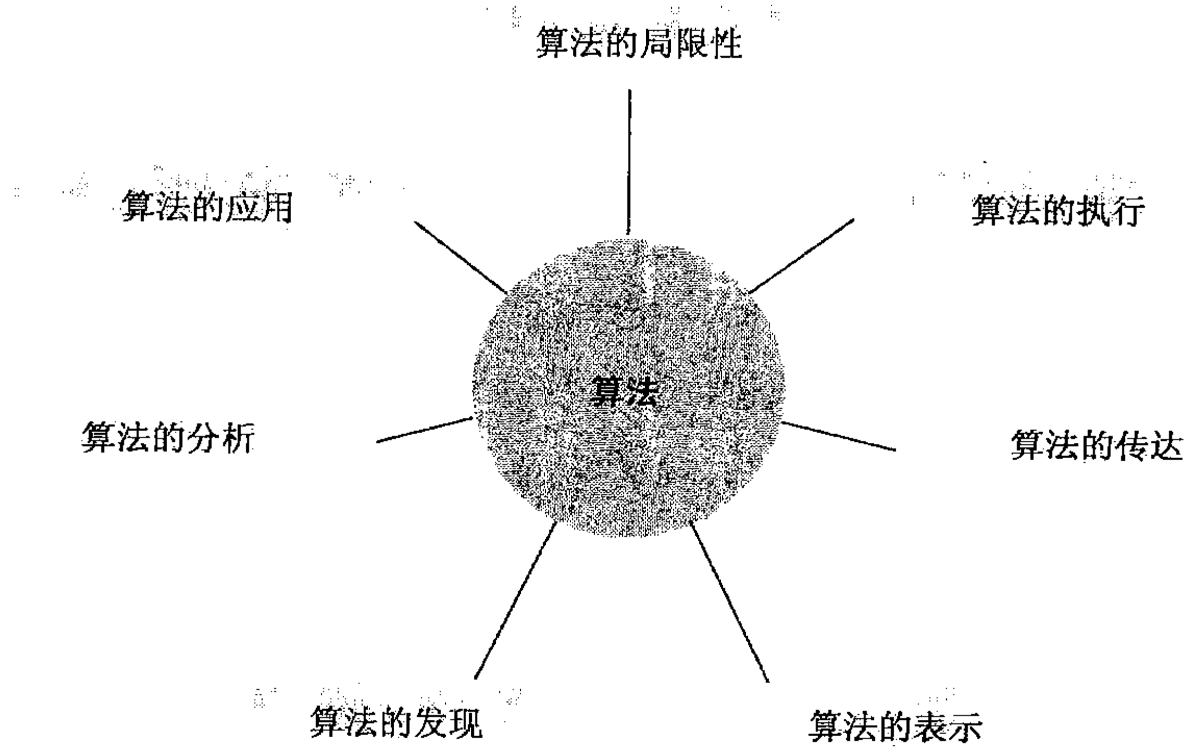


图0-5 算法在计算机科学中的核心地位

0.4 抽象

抽象概念贯穿计算机科学的研究和计算机系统的设计，因此有必要在绪论中简单介绍。**抽象**（abstraction）在本书中的意思是指一个实体的外部特征与其内部构成细节之间的分离。抽象使我们可以忽略一些复杂设备（如计算机、汽车和微波炉等）的内部细节，而把它们作为单一的可理解的单元，而且正是通过抽象，这些复杂的系统才能够被设计和生产出来。计算机、汽车和微波炉由若干部件构成，而这些部件又分别由更小的部件构成。每个部件表示一层抽象，在此层面上，该部件的使用与它内部构成细节是分隔的。

运用抽象，我们能够构造、分析和管理大型的复杂计算机系统，但如果从细节的层面上看问题，就会使人不识庐山真面目。在每一个抽象层面上，我们都把此系统看成是由若干称为**抽象工具**（abstract tool）的部件组成的，而暂时忽略这些部件的内部构成。这样我们的精力就集中了，可以考虑一个部件如何与同一层面其他部件发生作用，以及这些部件如何作为一个整体形成更高级别的部件。由此我们就可以理解该系统中与手头任务有关的那部分，而不会在众多的细节中迷失方向。

需要强调的是，抽象并不局限于科学和技术领域，它是一门重要的简化技术，我们的社会所形成的任何一种生活方式都离不开抽象。很少有人知道，日常生活中各种各样的便利是怎样实现的：我们需要吃饭穿衣，但却不能都自己生产；我们使用电器设备和通信系统，但不需要了解它们的内部技术；我们享受其他人提供的服务，但不需要知道他们的专业细节。对每一项新的发展只有一小部分社会成员专职于其实现，其他人则将实现的结果作为抽象工具来使用。这样，社会的抽象工具仓库扩大了，社会进一步发展的能力也增强了。

抽象这一话题在本书中会被反复提及。我们将了解到，计算设备是通过各种抽象工具构建的。我们还会看到，大型软件系统开发是以模块化方式完成的，其中每个模块都被作为较大模块上的一种抽象工具。此外，在计算机科学本身的发展中，抽象也扮演了很重要的角色，有了它，研究人员可以把精力集中在一个复杂领域中的特定范围。实际上，本书的编排也反映了该科学的这种特征——每一章都围绕着计算机科学一个特定的范围，而且往往出人意料地完全独立于其他各章，但所有这些章合在一起又形成了对该科学所涉及领域的全面介绍。

0.5 学习大纲

本书遵循自底向上的方法讲述计算机科学，先从读者有亲身体验的主题开始（比如计算机硬件），继而引出比较抽象的主题（比如算法复杂性和可计算性）。结果是我们的学习遵循了这样一个模式：随着对主题的深入理解，我们构建了规模越来越大的抽象工具。

我们首先学习的主题与设计和构造执行算法的机器有关。第1章（数据存储）学习现代计算机的信息编码和信息存储问题，第2章（数据操控）研究简单计算机的内部基本操作。虽然部分学习内容涉及技术问题，但总体上是独立于具体技术的。也就是说，像数字电路设计、数据编码与压缩系统，以及计算机体系结构这样的话题在更广阔的技术领域中都很重要，并且不管技术发展方向如何，它们的重要性都不会降低。

在第3章（操作系统）中，我们将学习控制一台计算机总体操作的软件，这种软件称为操作系统。操作系统控制计算机与其外部世界之间的接口：保护计算机及其内部所存储的数据，以免被非授权用户访问；允许计算机用户请求执行各种程序；协调内部活动，以满足用户请求。

在第4章（组网及因特网）中，我们将学习如何连接计算机以构成计算机网络，以及网络是如何连接成互联网的。由此引出诸如网络协议、因特网结构和内部操作、万维网，以及诸多安

全性问题等主题。

第5章（算法）比较规范地介绍了算法。我们要研究算法的发现，明确几种基本的算法结构，开发几项表示算法的初等技术，并介绍算法的有效性和正确性问题。

第6章（程序设计语言）研究的问题是算法表示和程序开发过程。我们会发现，人们在不断改善程序设计技术的过程中，已经创造出各种各样的程序设计方法学或范式，而每一种都有自己的一套程序设计语言。我们将研究这些范式和语言，以及语法和语言翻译的问题。

第7章（软件工程）介绍计算机科学的一个分支——软件工程。软件工程处理的是开发大型软件系统时所遇到的问题。大型软件系统的设计是一项复杂的任务，会遇到传统工程未涉及的许多问题。因此，软件工程这一学科已经成为计算机科学中一个重要的研究领域，从工程、项目管理、人力资源管理、编程语言设计乃至建筑学等学科中吸取了大量养分。

在接下来的两章中，我们将学习在计算机系统中组织数据的方法。第8章（数据抽象）介绍传统上用于在计算机主存储器中组织数据的技术，然后探索数据抽象的演变发展，从原语的概念一直到今天的面向对象式技术。第9章（数据库系统）介绍传统上用于在计算机海量存储器中组织数据的方法，并研究如何实现非常大的复杂数据库系统。

在第10章（计算机图形学）中，我们研究图形和动画，这是一个创建并图像化虚拟世界的领域。由于像机器体系结构、算法设计、数据结构和软件工程等计算机科学传统领域的发展，图形和动画学科取得了显著进展，业已发展成为激动人心、充满活力的学科。此外，这个领域说明了计算机科学的各个组成部分是如何与物理、艺术和摄影术等学科相结合以产生显著成果的。

在第11章（人工智能）中，我们将了解到，为了开发更有用的机器，计算机科学现已一马当先，转向研究人类智能，希望通过对我们自己的思维推理和认知的了解，能设计出模拟这些过程的算法，从而把这些能力传递给机器。结果是，计算机科学又诞生了一个称为人工智能的领域，它非常依赖于心理学、生物学和语言学等领域的研究。

我们的学习到第12章（计算理论）结束，在这一章中介绍了计算机科学的理论基础，这个主题使我们了解了算法（和机器）的局限性。在本章，我们不但明确了几个算法上不能解决的问题（它们在理论上也是超出机器能力的），而且认识到解决其他许多问题需要大量的时间或空间，以致从实践的角度上讲也是不可解的。因此，我们能够领悟算法系统的应用范围和局限性。

我们的目标是，每一章都在一定深度上使读者真正理解所讨论的主题。我们希望所阐述的计算机科学知识会对大家的工作有所帮助——使读者了解自己所生活的技术社会，打好跟随科技进步自我学习的基础。

0.6 社会影响

计算机科学的进步正淡化着许多差别，而这些差别正是我们过去作出某些决策的基准，而且计算机科学的进步也向社会的许多准则提出了挑战。在法律上，因此产生了某些疑问——知识产权的度以及伴随这个所有权的权利和义务。在伦理上，人们面临着许多挑战传统社会行为准则的抉择。对于政府，又产生了许多争议——计算机技术及其应用应该规范到什么程度？在哲学上，人们开始争论智能行为的存在与智能本身的存在。同时，整个社会也在讨论：新的计算机应用是代表新的自由还是新的控制？

虽然这些话题不属于计算机科学本身的范畴，但是对于那些想涉足计算机或者相关领域的人，它们还是很重要的。科学新发现经常会使许多应用产生争议，这使得人们对相关的研究人员产生极大不满。进一步而言，伦理上的过错足以摧毁本可以很成功的事业。

计算机技术的发展给人们提出了许多难题，而具备一些解决此类问题的能力对于非计算机领域的人也十分重要。的确，计算机技术已经在全社会迅速普及，几乎无人不受其影响。

本书提供了一些技术背景，有助于人们以一种理智的思维处理计算机科学所产生的问题。然而，计算机科学的技术知识本身无法提供全部问题的解决办法。因此，本书的一些章节致力于介绍社会、伦理和法律上的问题，包括安全性、软件所有权和义务问题、数据库技术的社会影响以及人工智能发展的后果。

此外，一个问题通常并不只有唯一一个正确的答案，许多有效的解决方案都是在对立的（也许都是有理的）观点之间进行折中的。因此，寻找解决方案通常需要这样的能力：能够倾听、辨别其他各种观点，开展理性的讨论，并在获得新的见解时改变自己的观点。于是，本书每章最后都有一系列“社会问题”，研究计算机科学和社会的关系。这些问题不是必须作答的，而是需要思考的。在许多情况下，一个乍看毫无疑问的答案等你发现其他可能性时就值得深思了。简言之，给出这些问题并不是要指导大家找到“正确”答案，而是要提高大家的意识，要意识到一个问题会牵扯多位利益相关者，一个问题会有多个解决方案，以及那些解决方案都同时具有长短期效应。

在结论最后，我们介绍了一些伦理学方法，这些方法是哲学家在基础理论的研究中提出的，从而产生了指导决策和行为的原则。这些理论大体可以归类为：结果伦理、职责伦理、合同伦理以及基于性格的伦理。你也许希望用这些理论处理本书中呈现的伦理问题。特别值得一提的是，你可能会发现不同的理论会导致相反的结论，从而将隐藏的候选方法呈现出来。

结果伦理试图基于作出各种选择所造成的后果分析问题。最突出的一个例子就是“功利主义”——“正确”的决策或行动可以带给社会上大多数人最大利益。乍一看，功利主义似乎很合理地解决了伦理上的难题，但是，它又会导致许多令人无法接受的后果。例如，他使少数人要服从多数人。此外，很多人认为伦理理论的结果方法太过强调结果，这样人就仅仅被当作实现结果的工具而不是有意义的个体了。他们还认为，这是所有结果伦理理论的一个最基本的缺陷。

和结果伦理相反，职责伦理并不考虑决策和行动的结果，它认为社会成员本身应该有职责或义务，因此又产生了需要解决的伦理问题。例如，一个人有尊重他人权利的义务，因此无论后果如何，他都要反对奴隶制。另外，反对职责伦理的人认为，对于有争议的职责问题，它无法提供解决方案。如果说出事实真相会使同事失去自信，你还会这样做吗？一个民族如果在战争中自卫，那么在随后的战争中就会牺牲很多公民，这个民族还应该自卫吗？

合同伦理理论首先假设社会没有任何伦理根基。在这种纯天然的背景下，什么情况都可能发生——每个人都必须自我保护，并不断防止他人的侵害。因此，合同伦理理论认为社会成员之间应该建立“合同”。例如，你不剽窃我，我就不剽窃你。进而，这些“合同”就成为伦理习惯的准绳。这里需要指出的是，合同伦理理论是伦理行为的动力，因为如果不遵循合同伦理我们就将生活得很不愉快。然而，反对合同伦理理论的人认为，它不能为伦理难题的解决提供足够广阔的基础，只有在那些已经建立合同的领域，它才能起到指导作用。（在没有合同约束的领域，我就可以为所欲为。）特别值得一提的是，新技术可能发现人们未知的领域，在其中无法应用现存的伦理合同。

性格伦理（有时称为德行伦理）是由柏拉图和亚里士多德提出的，它指的是“好行为”不是运用各种规则的结果，而是“良好性格”的自然结果。当一个人解决伦理难题时，结果伦理、职责伦理以及合同伦理认为应该考虑的分别是，“结果会怎样呢？”“我的职责是什么呢？”“我有什么合同可作为依据呢？”而性格伦理考虑的是，“我想成为什么样的人呢？”因此，好行为是建立在好性格基础上的，而这正得益于良好的教育以及德行习惯。

向不同专业领域人士教授伦理知识时，一般以性格伦理为基础，即不用教授专门的伦理理论，而是举一些案例，暴露专业领域的各种伦理问题。通过讨论这些案例的利弊，这些专业人士就会对职业生活中潜在的危险有一个更清醒、更深入和更敏感的认识了，并将这种认识融入到他们的性格中。这就是每章最后设计社会问题的精神所在。

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答出这些问题还不够，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 如果没有计算机革命，我们现在的社会将有很大不同。人们已经广泛接受这种观点。现在的社会是更好，还是更差？如果你在社会中的地位不同，答案是否也会不同？
2. 不努力了解技术的基础知识，却又想积极参与到当今的技术社会中，这种做法是否可行？例如，要通过表决来决定如何支持和使用某种技术，那么表决者是否有责任了解那种技术？你的答案是否与具体哪种技术有关？例如，考虑使用核技术时和考虑使用计算机技术时，回答是否一样？
3. 传统上人们选择现金交易方式处理账务，因而不需要支付服务费用。然而，我们经济生活中自动化程度在不断提高，金融机构对使用某些自动化系统收取服务费用。那么，“服务收费不公正地限制了人们参与经济活动”这种说法是否正确呢？例如，假设雇主仅用支票支付雇员的工资，并且所有金融结构都对支票兑现和存款收取服务费用，那么雇员是否因此受到了不公正的待遇呢？如果雇主坚持通过直接存款的方式支付工资，那该怎么办呢？
4. 在交互式电视节目中，某一个公司有可能会从孩子那里获取有关其家庭的信息（也许是通过交互式游戏），那应该控制到什么程度呢？例如，是否可以允许公司通过孩子得知其父母的购物习惯？那么关于孩子自己的信息呢？
5. 政府对计算机技术及其应用的法规管制应当到什么程度？例如，考虑一下问题3和问题4中提到的问题。政府管制的依据是什么？
6. 关于技术，尤其是计算机技术，我们所作出的决策会对我们的后代有多大的影响？
7. 随着技术的进步，我们的教育系统不断面临挑战，要重新考虑科目安排的抽象层次。许多问题是类似的，如某项技能是否必要，是否允许学生依赖某种抽象工具等。学三角时，不再教学生如何利用函数表求三角函数的值，而是允许学生用计算器作为抽象工具来求函数值。有些人认为，长除也应该让位于抽象。还有哪些主题涉及类似的争论？现代的文字处理软件是否会使人们不需要练习书法？视频技术的使用是否会在将来的某一天取代阅读？
8. 所有公民都有权获得信息，因而才设立了那么多公共图书馆。越来越多的信息通过计算机技术存储和传播，是否每一位公民都应该有权利访问这个技术系统呢？答案如果是肯定的，那么公共图书馆是否应该为这种访问提供渠道呢？
9. 在一个依靠抽象工具的社会里，会产生什么样的伦理问题呢？是否存在这样的情况，当我们使用某个产品或某项服务时，不了解它们的工作原理，不了解其生产方法就有悖道德？亦或不了解使用它会带来的副作用就有悖道德？
10. 随着我们社会的逐步自动化，政府监督公民的活动变得很容易。这是好还是坏呢？
11. George Orwell在他的小说《1984》中想象的哪些技术已经实现？它们的使用方法是否与Orwell预想的一样？

12. 如果你有一台时间机器，你想生活在哪一个历史阶段？有你想带去的现代技术吗？你所选择的技术可以脱离其他技术而被你单独带走吗？一项技术可以在多大程度上独立于其他技术？防止温室效应，却又接受现代医疗，这两者相符吗？
13. 假如由于工作关系，你必须生活在另一种文化氛围中。你会按照自己的本土文化习惯我行我素，还是会选择遵循所在地的异域生活习俗？对这个问题的回答，是否因为跟穿衣打扮还是人权有关而不同呢？如果你是在本国生活，但需要处理各种外来文化冲突，那你会坚持什么道德标准？
14. 在商务、通信和社交互动方面，社会是否已太过依赖于计算机应用？例如，如果长期中断因特网或移动电话服务，会有什么后果？
15. 大多数智能手机都能够利用GPS识别手机的位置。这样一来，相关的应用程序就可以基于手机的当前位置提供与该位置相关的信息（如本地新闻、本地天气，或者附近的商业机构）。然而，这些GPS功能却也可能支持其他应用将手机的位置广播给其他各方。这样好吗？手机的位置（继而手机用户的位置）信息会被如何滥用呢？
16. 根据你对以上问题的回答，你打算支持0.6节中的哪一个伦理理论？

课外阅读

Goldstine, J. J. *The Computer from Pascal to von Neumann*. Princeton: Princeton University Press, 1972.

Kizza, J. M. *Ethical and Social Issues in the Information Age*, 3 rd ed. London: Springer-Verlag, 2007.

Mollenhoff, C. R. *Atanasoff: Forgotten Father of the Computer*. Ames: Iowa State University Press, 1988.

Neumann, P. G. *Computer Related Risks*. Boston, MA: Addison-Wesley, 1995.

Ni, L. *Smart Phone and Next Generation Mobile Computing*. San Francisco: Morgan Kaufrmann, 2006.

Quinn, M. J. *Ethics for the Information Age*, 2nd ed. Boston, MA: Addison-Wesley, 2006.

Randell, B. *The Origins of Digital Computers*, 3 rd ed. New York: Springer-Verlag, 1982.

Spinello, R. A. and H. T. Tavani. *Readings in CyberEthics*, 2nd ed. Sudbury, MA: Jones and Bartlett, 2004.

Swade, D. *The Difference Engine*. New York: Viking, 2000.

Tavani, H. T. *Ethics and Technology: Ethical Issues in an Age of Information and Communication Technology*, 3rd ed. New York: Wiley, 2011.

Woolley, B. *The Bride of Science, Romance, Reason, and Byron's Daughter*. New York: McGraw-Hill, 1999.

第1章

数据存储

在本章中，我们学习有关计算机中数据表示和数据存储的内容。我们要研究的数据类型包括文本、数值、图像、音频和视频。除了传统计算外，本章的很多内容还涉及数字摄影、音频/视频录制和复制，以及远程通信等领域。

本章内容

我们首先要学习的是在计算机科学中信息如何编码和存储。第一步，我们要讨论计算机数据存储设备的基础知识，然后研究如何进行信息编码并将其存储到系统内部。我们还将探讨现如今数据存储系统的各个分支，以及如何用数据压缩、错误处理等技术来克服其不足。

1.1 位和位存储

在今天的计算机中，信息是以0和1的模式编码的。这些数字称为**位**（bit，binary digits的缩写）。尽管你可能倾向于把它们与数值联系在一起，但它们的确只是些符号，其意义取决于正在处理的应用：有时用来表示数值；有时又代表字母表里的字符和标点符号；有时表示图像；有时还表示声音。

1.1.1 布尔运算

为了理解单独的位在计算机中是如何进行存储和操作的，这里我们假设位0代表FALSE（假），位1代表TRUE（真），这样表示就可以把对位的运算看做是对真/假值的操作。数学家乔治·布尔（George Boole，1815—1864）是逻辑数学领域的先驱，为了纪念他，人们把处理真/假值的运算命名为**布尔运算**（Boolean operation）。3个基本的布尔运算是AND（与）、OR（或）以及XOR（异或），见图1-1。这些运算类似于算术运算的乘法和加法，因为它们结合一对值（运算输入），然后得出第三个值（运算输出）。不过，与算术运算不同的是布尔运算结合的是真/假值，而不是数值。

布尔运算AND是用于反映由两个较小、较简单语句通过连接词AND组成的语句的真/假值。一般形式如下：

P AND Q

其中，P代表一个语句，Q代表另外一个语句。例如：

Kermit是一只青蛙 AND Piggy小姐是一位演员

AND运算的输入是复合语句分句的真/假值，输出则是复合语句本身的真/假值。因为P AND Q语句的值只有在其两个分句都是真时才为真，所以可以得出结论：1 AND 1的输出是1，而其他所有情况的输出值都将是0，如图1-1所示。

同理，OR运算是基于如下形式的复合语句：

P OR Q

同样，P代表一个语句，Q代表另外一个语句。当其中至少有一个分句为真时，语句才为真，见图1-1。

AND运算

0 AND 0 = 0	0 AND 1 = 0	1 AND 0 = 0	1 AND 1 = 1

OR运算

0 OR 0 = 0	0 OR 1 = 1	1 OR 0 = 1	1 OR 1 = 1

XOR运算

0 XOR 0 = 0	0 XOR 1 = 1	1 XOR 0 = 1	1 XOR 1 = 0

图1-1　布尔运算AND、OR和XOR

英语中没有一个连词可以单独表示XOR。当两个分句一个为1（真），另一个为0（假）时，此XOR运算值是真。例如，P XOR Q语句的意思是“或者是P，或者是Q，但不会是两个共存”。（简言之，当两分句不同时，XOR运算为真。）

NOT（非）运算是另一个布尔运算。它区别于AND、OR和XOR，因为它只有一个输入。它的输出就是输入值的相反值。如果NOT运算的输入值是真，那么它的输出值为假，反之亦然。因此，如果NOT运算的输入是下面的语句的真/假值：

Fozzie is a bear.

那么，其输出就是如下语句的真/假值：

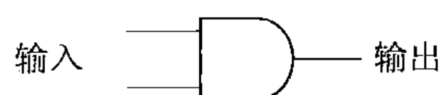
Fozzie is not a bear.

1.1.2　门和触发器

门（gate）指的是一种设备，给出一种布尔运算输入值时，可以得出该布尔运算的输出值。门可以通过很多种技术制造出来，如齿轮、继电器和光学设备。今天的计算机中，门经常是通过微电子电路实现的，其中数字0和1由电压电平表示。不过，我们不需要关注这些细节问题。

对于我们来说，知道用符号形式来表示门就足够了，如图1-2所示。注意，与、或、异或及非门是分别用不同形状的图表示的，输入值在一边，输出值在另一边。

与门



输入	输出
0 0	0
0 1	0
1 0	0
1 1	1

或门



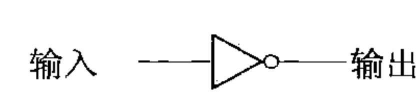
输入	输出
0 0	0
0 1	1
1 0	1
1 1	1

异或门



输入	输出
0 0	0
0 1	1
1 0	1
1 1	0

非门



输入	输出
0	1
1	0

图1-2　与、或、异或以及非门的图例及输入和输出值

这样的门为构造计算机提供了基础构件。构造计算机时，图1-3所示的电路是一个重要的环节，该电路是一个称为触发器的电路特例。**触发器**（flip-flop）是一个可以产生0或1输出值的电路，它的值会一直保持不变，除非其他电路过来的临时脉冲使其改变成另一个值。换句话说，输出值在外界的刺激下在两个值之间相互转换。如图1-3所示，只要电路两个输入值一直都是0，那么输出值（无论是0还是1）就不会改变。不过，如果在它的上输入端临时放置一个1，那么将强制其输出值为1；反之，在它下输入端临时放置一个1，那么将强制其输出值为0。

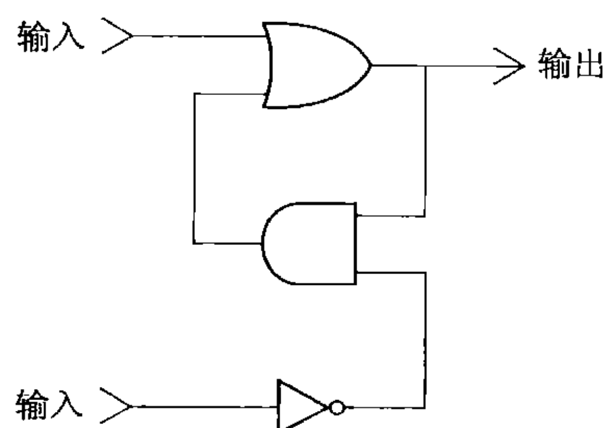
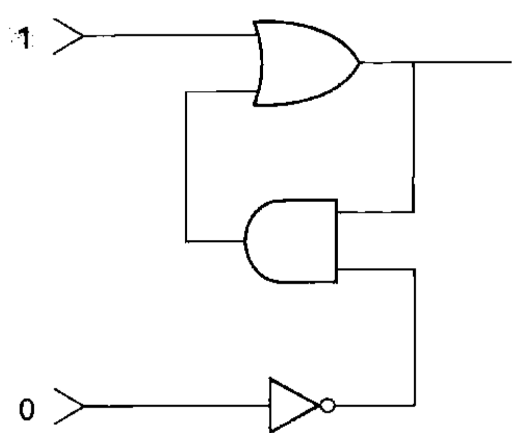


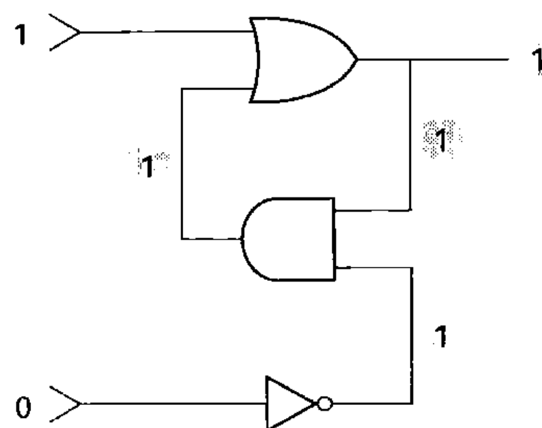
图1-3　一个简单的触发器电路

我们来仔细研究一下这个问题。在我们不知道图1-3中电路的当前输出值的情况下，假设上面的输入值变为1，而下面的输入值仍为0（见图1-4a)，那么不管这个门另外一个输入值是什么，

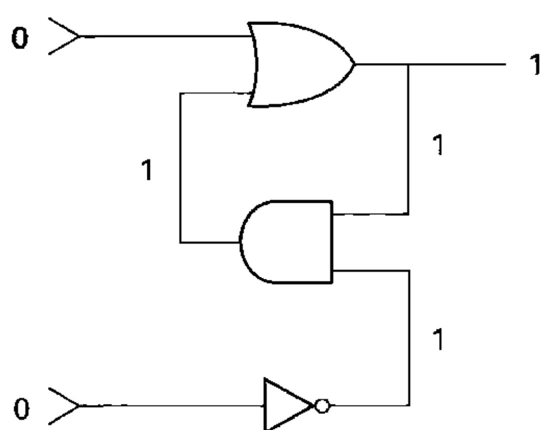
或门的输出值都将为1。这时，与门的两个输入值都为1，因为这个门的另外一个输入值已经为1（由经过触发器下输入端的非门获得）。与门的输出值于是变成1，也就意味着现在或门的第二次输入值将为1（见图1-4b）。这样就可以确保即使触发器上面的输入值变回0（见图1-4c），或门的输出值也会保持为1。总之，触发器的输出值已经为1，那么输入值变回0时，其输出值仍然保持不变。



(a) 将上面的输入置1



(b) 这使或门的输出为1，接着使与门的输出为1



(c) 将上面的输入变为0之后，由于与门的输出为1，所以或门的输出仍然为1

图1-4　将一个触发器的输出值设置为1

同理，在下输入端上临时放置数值1会强制触发器的输出值为0，而且输入值变回0时，输出值仍然保持不变。

我们介绍触发器电路（见图1-3和图1-4）有3个原因。首先，它向我们展示了设备是如何通过门制造出来的，这是一个数字电路的设计过程，在计算机工程领域是一个很重要的课题。事实上，在计算机工程中，触发器只是诸多基础工具电路的一种。

第二，触发器的概念为抽象和使用抽象工具提供了一个例子。事实上，可以用多种方法设计触发器。图1-5给出了其中的一种方法，如果你用这个电路做实验就会发现，尽管它有着不同的内部结构，但它与图1-3中的外部特性是一样的。计算机工程师不必知晓触发器中实际使用的是哪种电路，只需理解触发器的外部特性并将其作为一个抽象工具来使用即可。一个触发器和其他定义良好的电路形成了一组构建块，而工程师就用这些构建块构造更复杂的电路。这样，计算机电路的设计就会呈现一种层次结构，其中每一层都将较低层次的构件作为抽象工具使用。

介绍触发器的第三个目的在于，触发器是在现代计算机中存储二进制位的一种方法。更精确地说，触发器可以被设置为具有0或1的输出值。其他电路可以通过发送脉冲到触发器的输入端调整这个值，还有其他一些电路可以将触发器的输出作为它们的输入来响应存储的值。这样，许多触发器被构造成非常小的电子电路，可以用在计算机内作为记录信息的一种方法，这些信息被编码成0和1的模式。实际上，众所周知的VLSI（Very Large-Scale Integration，超大规模集

成）技术支持将成千上万个电子元件构造在一个晶片［称为**芯片**（chip）］上，它用来创建在控制电路中含有成千上万个触发器的微型设备。然后，这些芯片被用作构建计算机系统的抽象工具。事实上，在某些情况下VLSI被用来在单块芯片上创建整个计算机系统。

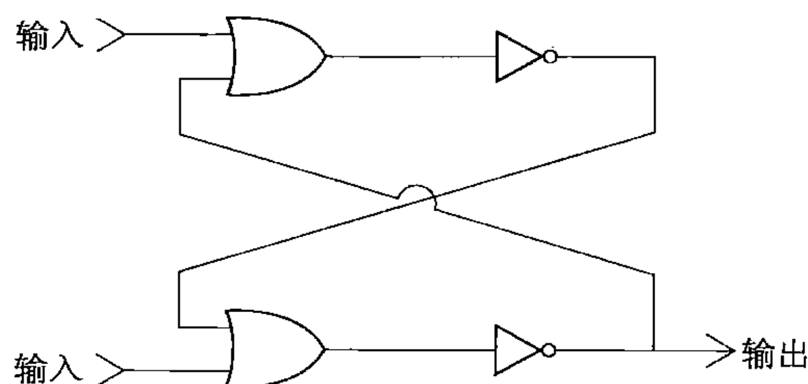


图1-5 构建触发器的另一种方法

1.1.3 十六进制记数法

当考虑计算机内部活动时，我们必须和位串打交道，有一些位串会非常长。一个长的位串常被称为**流**（stream）。但是，人脑不容易理解流。仅仅抄录位模式的101101010011就很乏味且容易出错误。因此，为了简化这种位模式的表示方法，我们常使用一种称为**十六进制记数法**（hexadecimal notation）的简写符号，它是利用计算机位模式的长度为4的倍数这样一个事实制定的。具体来说就是，十六进制记数法用一个符号表示位模式的4位。例如，一个12位串只需要3个符号就可以表示。

图1-6呈现了十六进制编码系统。左边一列展示的是所有长度为4的位模式，右边一列展示的是十六进制中代表左边位模式的符号。通过这个系统，10110101形式表示为B5。这是通过把位模式拆分为长度为4的子串，然后用十六进制的符号代替每一个子串得到的——1011由B来表示，0101由5来表示。同理，十六位模式1010010011001000可以缩减成更易为人接受的形式A4C8。

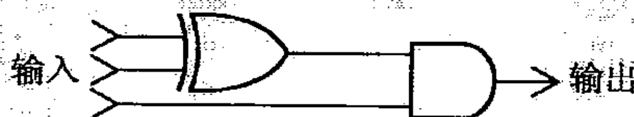
位模式	十六进制表示
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

图1-6 十六进制编码系统

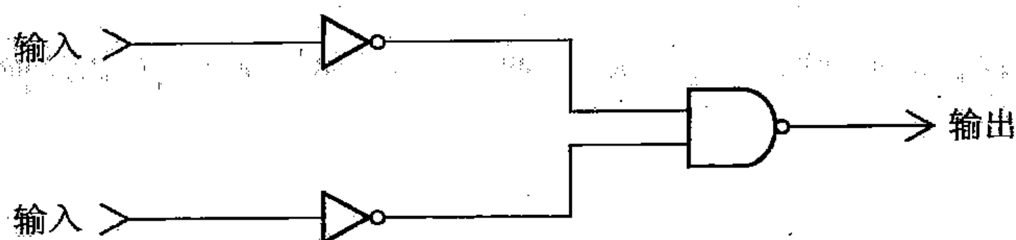
第2章将广泛使用十六进制记数法，由此你就能体会到它的效率。

问题与练习

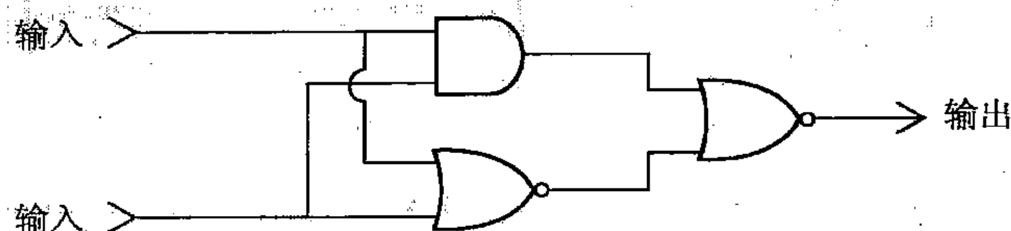
1. 什么样的位模式输入可以使得下面的电路输出值为1？



2. 对于图1-3中的触发器，我们在文中强调，下输入端放置1（保持上输入端为0），这样就迫使触发器的输出为0。描述一下这种情况触发器内部的活动序列。
3. 假定图1-5中触发器的输入都为0，描述一下当上输入端被临时设为1时所发生的活动序列。
4. a. 如果一个与门的输出值传递给了一个非门，那么这个组合电路计算的布尔运算称为与非。当且仅当输入值都为1时，输出值为0。与非门的符号和与门的符号类似，只是输出有一个圆圈。下面的电路包含与非门，那么这个电路完成什么布尔运算？



b. 如果一个或门的输出值传递给一个非门，那么这个组合电路计算的布尔运算称为或非，当且仅当输入值都为0时，输出值为1。或非门的符号和或门的符号类似，只是输出有一个圆圈。下面的电路包含一个与门和两个或非门。那么这个电路计算称为什么布尔运算？



5. 用十六进制记数法来表示下面的位模式。

a. 0110101011110010　b. 111010000101010100010111　c. 01001000

6. 下面的十六进制模式表示什么位模式？

a. 5FD97　b. 610A　c. ABCD　d. 0100

1.2 主存储器

为了存储数据，计算机包含大量的电路（如触发器），每一个电路能够存储单独的一个位。这种位存储器被称为计算机的**主存储器**（main memory）。

1.2.1 存储器结构

计算机主存储器是以称为**存储单元**（cell）的可管理单位组织起来的，一个典型的存储单元容量是8位。[一个8位的串称为一个**字节**（byte），因此一个典型的存储单元容量是一个字节。]在像微波炉这样的家用电器中所使用的小型计算机的主存储器，仅仅包含几百个存储单元，但是大型计算机的主存储器可能有上亿个存储单元。

虽然计算机中没有左或右的概念，但是我们通常假设存储单元的位是排成一行的。该行的左端称为**高位端**（high-order end），右端称为**低位端**（low-order end）。高位端的最左一位称作高位或**最高有效位**（most significant bit）。取这个名称是因为，如果把存储单元里的内容解释为数值，那么这一位就是该数的最高有效数字。类似地，低位端的最右一位称为低位或**最低有效位**（least significant bit）。于是，我们可以如图1-7所示的那样描述字节型存储单元的内容。

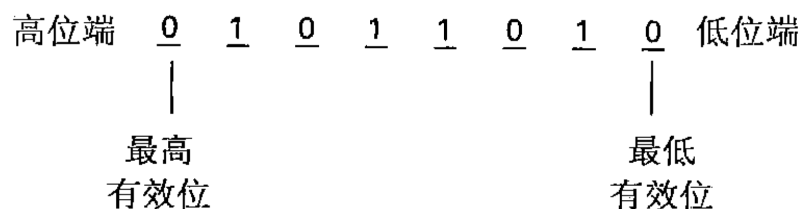


图1-7 字节型存储单元的结构

为了区分计算机主存储器中的各存储单元，每一个存储单元都被赋予了一个唯一的“名字”，称为**地址**（address）。这类似于通过地址找到城市里的一座座房屋。不过，对于存储单元，所用地址都是用数字表示的。更精确地说，我们把所有的存储单元都看做是排成一行的，并按照这个顺序从0开始编号。这样的编址系统不仅为我们提供了唯一标识每个存储单元的方法，而且也

给存储单元赋予了顺序的概念（见图1-8），这样就有了诸如“下一个单元”、“前一个单元”的说法。

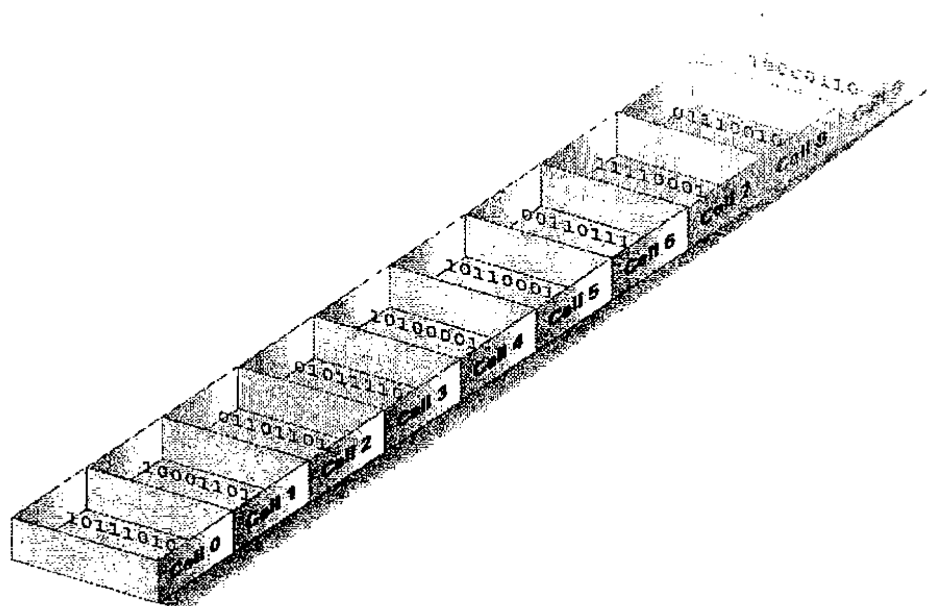


图1-8 按地址排列的存储单元

将主存储器的存储单元和存储单元的位都进行排序，就产生一个重要结果，即计算机主存储器的所有二进制位本质上被排成了一长行，因而这个长行上的片段就可以存储比单个存储单元要长的位模式。特别是，我们只需要两个连续的存储单元就可以存储16位的串。

为了做成一台计算机的主存储器，实际存放二进制位的电路还组合了其他的电路，这些电路使得其他电路可以在存储单元中存入和取出数据。以这种方式，其他电路可以通过电信号请求从存储器中得到指定地址的内容（称为读操作），或者请求把某个位模式存放到指定地址的存储单元里（称为写操作）。

因为计算机的主存储器由独立的、可编址的存储单元组成，所以可以根据需要独立访问这些存储单元。为了反映用任何顺序访问存储单元的能力，计算机的主存储器常被称为RAM（Random Access Memory，随机存取存储器）。主存储器的这种随机存取特性与1.3节中将要讨论的海量存储系统形成鲜明对比，在海量存储系统中长二进制串被作为合并块来操控。

尽管我们介绍说触发器可以作为二进制位的一种存储方法，但是在现代的大多数计算机中，RAM都是用其他可以提供更小型化和更快响应时间的技术制造的，其中许多技术将位存储为可快速消散的电荷。因此，这些设备需要附加的电路，称为刷新电路，可以在1 s内反复补充电荷很多次。因为它的这种不稳定性，所以通过这种技术构造的计算机存储器常被称为**动态存储器**（dynamic memory），于是就产生了术语**DRAM**（读作“DEE-ram”），用来表示动态RAM。有时候关于动态存储器也会用**SDRAM**（读作“ES-DEE-ram”），用来表示同步动态RAM，采取这种附加的技术可以缩短从存储单元取出信息所需要的时间。

1.2.2 存储器容量的度量

在第2章会学到，如果主存储器中存储单元的总数是2的幂，那么设计起来是很方便的，因此早期计算机存储器的大小通常以1024（2^{10}）个存储单元为度量单位。因为1024接近于数值1000，所以计算机行业的许多人采用前缀千（kilo）来表示这个单位。也就是说，术语**千字节**（kilobyte，简写形式为KB）用于表示1024字节。因此有4096个存储单元的计算机被称为有4 KB存储器（4096=4×1024）。随着存储器容量的增大，又新增了一些类似的度量单位，包括MB（兆字节）、GB（吉字节）、TB（太字节）。遗憾的是，这种前缀用法属于术语的误用，因

为这些前缀已经是其他领域用于指称1000的幂。例如，在度量距离时千米（kilometer）指的是1000米（m），在度量无线电频率时，兆赫（megahertz）指的是1 000 000赫兹（Hz）。在此提醒大家：一般说来，千（kilo-）、兆（mega-）等术语在涉及计算机存储器时表示2的幂，但在其他环境中表示1000的幂。

问题与练习

1. 如果地址为5的存储单元存有值8，那么在将值5写入6号存储单元和将5号存储单元的内容移到6号存储单元之间有什么差别？
2. 假定你想交换存储在2号和3号存储单元中的值。那么下面的步骤错在哪里？
   步骤1：把2号存储单元中的内容移到3号存储单元。
   步骤2：把3号存储单元中的内容移到2号存储单元。
   请设计能够正确交换这两个存储单元内容的步骤。如有必要可以使用额外的存储单元。
3. 4 KB计算机存储器里有多少个二进制位？

1.3 海量存储器

由于计算机主存储器的不稳定性和容量的限制，大多数计算机都有称为**海量存储**（mass storage）系统的附加存储设备，包括磁盘、CD盘、DVD盘、磁带、闪存驱动器（所有这些我们稍后会讨论）。相对于主存储器，海量存储系统的优点是更稳定、容量大、价格低，并且在许多情况下可以针对存档的需要从计算机上方便地取下这类存储设备。

术语**联机**（on-line）和**脱机**（off-line）通常分别用来描述那些既能接入计算机又能从计算机上移除的设备。**联机**意味着设备或信息已经与计算机连接，不需要人的干预就可以使用。**脱机**意味着必须先有人的干预，设备和信息才可被计算机使用——或许需要先打开这个设备，或许需要将包含该信息的介质插到某机械装置里。

海量存储系统的主要不足之处是，它们一般都需要机械运动。因为主存储器的所有工作都是由电子器件实现的，所以比起计算机主存储器来，海量存储系统的数据存取需要花费更长的时间。

1.3.1 磁学系统

很多年以来，磁技术已经占据了海量存储领域。最常见的例子便是我们今天使用的**磁盘**（magnetic disk），它里面是薄的、可以旋转的盘片，表面有磁介质的涂层用以存储数据（图1-9）。读/写磁头安装在盘片的上面和（或）下面，当盘片旋转时，每个磁头在盘片上面或下面相对于称为**磁道**（track）的圆圈转动。移动磁头时，可以对各个同心的磁道进行存取。在很多情况下，一个磁盘存储系统包含若干个安装在同一根轴上的盘片，层叠在一起，盘片之间留有足够的距离，使得磁头可以在盘片之间滑动。这种情况下，所有的磁头是一起移动的。因此，每当磁头移到新的位置时，就可以访问新的一组磁道，称为**柱面**（cylinder）。

因为一个磁道可以包含的数据通常比我们每一次要处理的数据多，所以每个磁道又被划分成若干个小弧区，称为**扇区**（sector）。记录在每个扇区上的信息是连续的二进制位串。磁盘上所有的扇区包含相同数目的二进制位（典型的容量是512个字节到若干KB），而且在最简单的磁盘存储系统里，每一个磁道被分为相同数目的扇区。因此，盘片边缘磁道扇区上存储的位密度要小于靠近盘片中心磁道上存储的位，这是因为外磁道要长于内磁道。事实上，在大容量磁盘

存储器系统里，边缘磁道可包含的扇区要远多于靠近中心的磁道，这种存储能力常通过一种称作ZBR（zoned-bit recording，区位记录）的技术得以应用。运用ZBR，一些相邻的磁道被统一命名为区，一个典型的盘片大约包含10个区。一个区的所有磁道有相同数目的扇区，但是靠外的区中每一个磁道包含的扇区比靠内的区包含的多。因此，盘片边缘的存储空间利用率要高了。不考虑细节，我们只需知道，一个磁盘存储系统包含许多独立的扇区，每一个扇区又可以作为独立的位串进行存取。

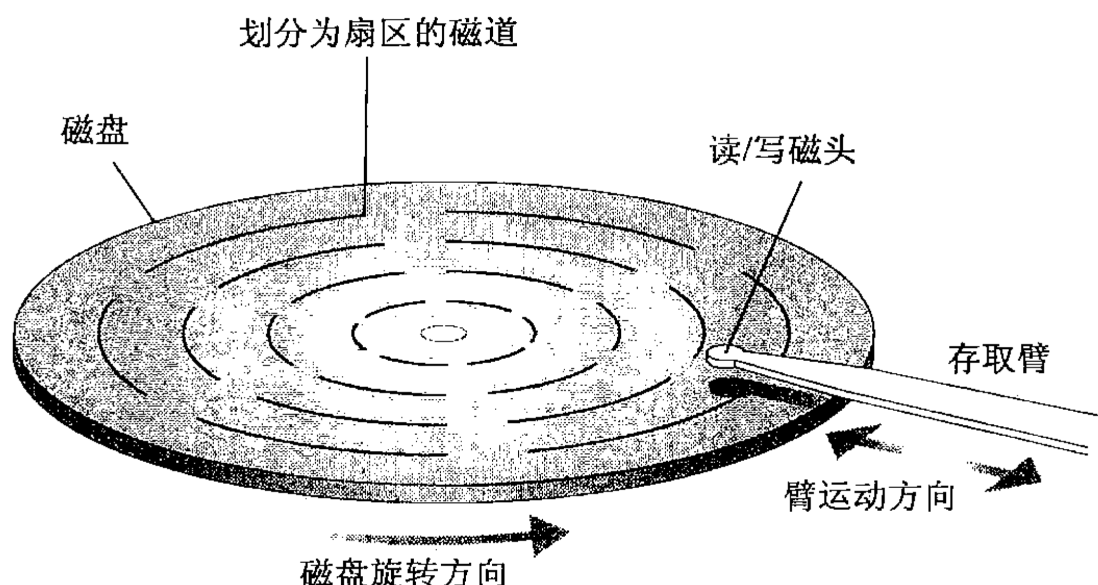


图1-9 磁盘存储系统

磁道和扇区的位置不是磁盘物理结构的固定部分，而是通过称为磁盘**格式化**（formatting）的过程磁化形成的。这个过程通常是由磁盘的厂家完成的，出厂的此类盘称为格式化盘。大多数计算机系统都能够执行此项任务。所以，如果一个磁盘的格式化信息被破坏了，那么可以重新格式化这个磁盘，不过这种操作将会扔掉原先记录在磁盘上的所有信息。

一个磁盘存储系统的容量取决于所用盘片数目以及所划分磁道与扇区的密度。较小容量的系统可能只有一个盘片。大容量磁盘系统的容量可达数GB，甚至TB，可能在同一根轴上安装有3～6个盘片。此外，数据有可能存储在每个盘片上下两面。

有几个标准可以用来评估一个磁盘系统的性能：（1）**寻道时间**（seek time），读/写磁头从一个磁道移到另一个磁道所需要的时间；（2）**旋转延迟**（rotation delay）或**等待时间**（latency time），盘片旋转一周所需要时间的一半，也就是读/写磁头到达所要求磁道后，等待盘片旋转使读/写磁头位于所要存取的数据（扇区）上所需要的平均时间；（3）**存取时间**（access time），即寻道时间和等待时间之和；（4）**传输速率**（transfer rate），在磁盘上读出或写入数据的速率。需要注意的是，在区位记录存储情况下，盘片旋转一次边缘道通过读/写磁头传递的数据要多于内区道，因此数据传输速率依所使用盘片部分的不同而有所变化。

限制磁盘存取时间和传输速率的一个因素是磁盘系统旋转的速度。为了支持高速旋转，这些系统里的读/写磁头不接触盘片，而只是“悬浮”在盘片表面。磁头与盘片之间的空间很小，以至于一粒小小的灰尘都可能卡在其中，并因此损坏磁盘和磁头，这一现象便是磁头划伤（head crash）。因此，磁盘系统出厂时都密封在箱子里。凭借这样的构造，磁盘系统能够以每秒几千次的速度旋转，达到每秒数以MB的传输速率。

因为磁盘系统的操作需要物理运动，所以难以与电子电路的速度相比。电子电路延迟时间是以纳秒（十亿分之一秒）甚至更小计算的，而磁盘系统的寻道时间、等待时间和存取时间是以毫秒（千分之一秒）度量的。因此，与电子电路等待结果的时间相比，从磁盘系统检索信息所需要的时间非常长。

磁盘存储系统不是唯一应用磁技术的海量存储设备。一种更古老的形式是**磁带**（magnetic

tape）（见图1-10），在这些系统里，信息存储在一条细薄的塑料带的磁涂层上，而塑料带则绕在磁带卷轴上作为存储器。为了存取数据，磁带应装到称为磁带驱动器的设备里，并可以在计算机控制下读带、写带和倒带。磁带驱动器有大有小，小至盒式机，大至比较老式的大型盘式机。盒式机又称为流式磁带机，磁带的外表与立体声收音机类似。虽然这些磁带机的存储容量依赖于所使用的格式，但是大多数都能达到几GB。

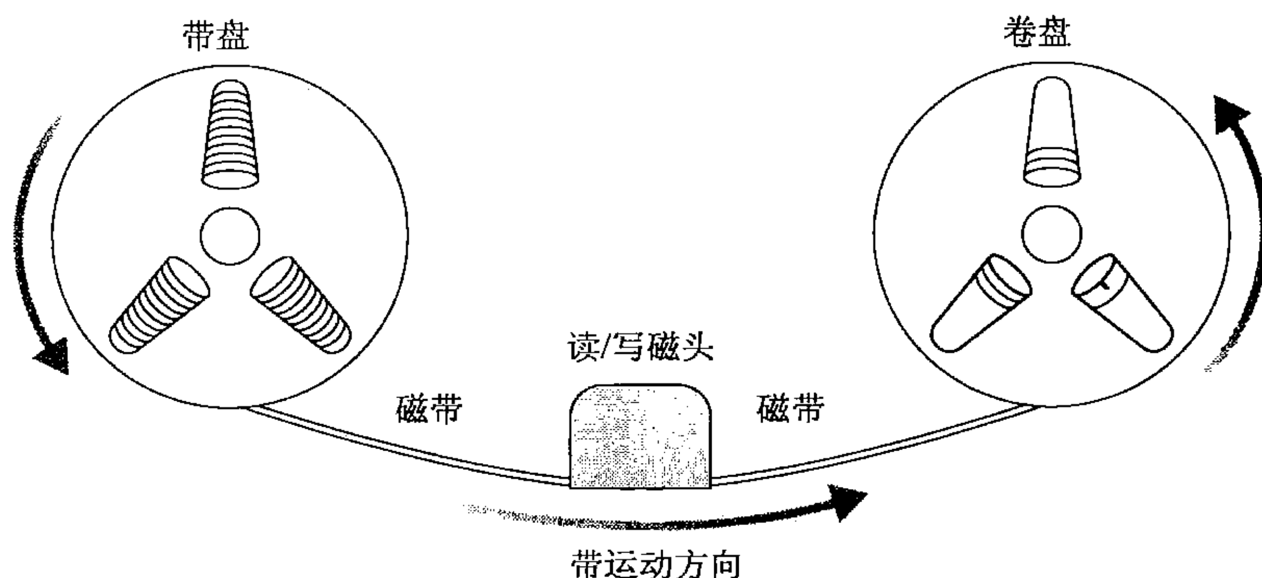


图1-10　磁带存储装置

磁带的一个主要缺点是，在磁带卷轴之间要移动的带子很长时，在一条磁带不同位置之间移动非常耗费时间。于是相对于磁盘系统而言，磁带系统的存取时间比较长，因为磁盘的读/写磁头只需要做短的移动就可以在不同的扇区存取数据。因此，磁带机对于联机的数据存储不是很常用。但是，磁带技术常应用在脱机档案数据存储中，原因是它具有容量大、可靠性高和性价比好等优势，但其他技术（如DVD、闪存等）的进步，正迅速地吞噬磁带系统最后的阵地。

1.3.2　光学系统

另一类海量存储器所应用的是光学技术，CD（Compact Disk，光盘）就是其中的一种。光盘的直径为12 cm（大约5英寸），由涂着光洁保护层的反射材料制成。通过在反射层上创建偏差的方法在光盘上面记录信息。激光束通过监视CD快速旋转时反射层的不规则反射偏差来读取信息。

CD技术最初用于音频录制，使用称为CD-DA（Compact Disk-Digital Audio，数字音频光盘）的记录格式，而今天CD作为计算机的数据存储设备，实质上使用的仍是同样的格式。特别值得一提的是，CD上的信息存储在一条磁道上，它呈螺旋形缠绕在CD上，很像老式唱片里的凹槽，不过与老式唱片不同的是，CD上的磁道是由内至外的（见图1-11）。这条磁道被划分为称为扇区的单元，每个扇区都有自己的标识，数据存储容量为2 KB，相当于在音频录制时$\frac{1}{75}$ s的音乐。

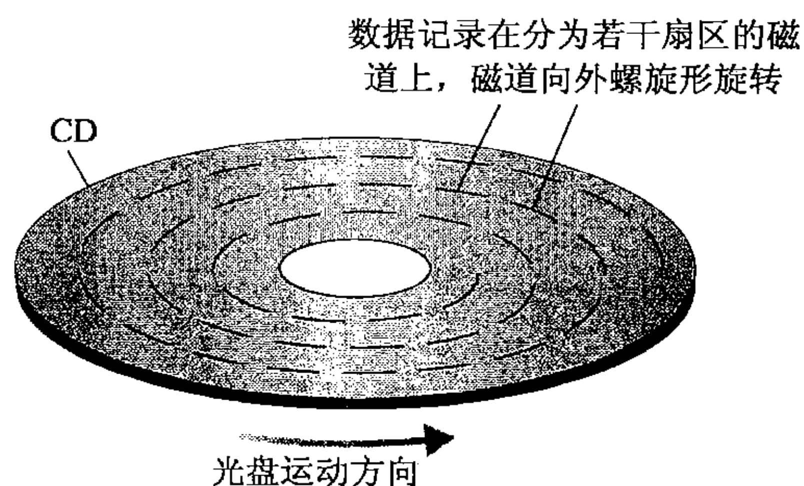


图1-11　CD存储格式

需要注意的是，盘片外部边缘的螺旋磁道距离比内部磁道距离要长。为了使CD的存储能力达到最大，信息就按照统一的线性密度存储在整个螺旋形磁道上。这就意味着，螺旋形磁道上靠外边缘的环道存放的信息比内部的环道多。所以，如果盘片旋转一整圈，激光束在扫描螺旋形磁道外面的部分时读到的扇区个数要比里面的部分多。因此，为了获得统一的数据传输速率，根据激光束在盘片上的位置，CD-DA播放器能够调整盘片的旋转速度。但是用于计算机数据存储的大多数CD系统，盘片旋转的速度是比较迅速和恒定的，因此其CD驱动器必须适应数据传输速率的变化。

由于采用这种设计思想，CD存储系统在处理长且连续的数据串（如音乐复制等）时表现最好。但是，当一个应用需要随机存取数据项时，磁盘存储器所用的方法（单个、同心磁道被划分成独立存取扇区的形式）就优于CD所用的螺旋形方法。

传统CD的存储容量是600～700 MB。但是，DVD（Digital Versatile Disk）[①]可具有多达几个GB的存储容量，它由多个半透明的层面构成，精确聚焦的激光可以识别其不同的层面。这种盘片能够存储冗长的多媒体信息，包括完整的一部电影。最后，蓝光技术（Blu-ray technology）使用蓝色（而非红色）激光，能够极为精确地聚焦激光束。因此，BD（Blu-ray Disk，蓝光光碟）的容量是DVD的5倍多。为了满足高清视频的需要，我们需要使用这种容量很大的存储设备。

1.3.3 闪存驱动器

基于磁学和光学技术的海量存储系统的一个普遍特征是，通过物理运动来存储和读取信息，例如，旋转磁盘、移动读/写磁头和扫描激光束等。这就意味着，数据存储和读取的速度比电子电路的速度要慢。**闪存**（flash memory）技术有潜力克服这个缺点。在一个闪存系统里，用电子信号将二进制位直接送到存储介质中，电子信号使得该介质中二氧化硅的微小晶格截获电子，从而转换微电子电路的性质。因为这些微小晶格能够保持截获的电子很多年，所以闪存技术适合存储脱机数据。

尽管存储在闪存系统里的数据能够像在RAM应用中一样，以小字节单元存取，但是现代技术规定存储的数据应批量擦除。不过反复的擦除会逐渐损坏二氧化硅的晶格，这就意味着现今的闪存技术不适合主存储器应用，主存储器的内容在一秒钟可能被改变许多次。然而，在某些应用里改变可以被控制在一个合理的水平，例如数码相机、移动电话、手持式PDA，所以闪存已经成为海量存储技术的一个选择。的确，因为闪存对物理震动不敏感（与磁学系统和光学系统不同），它在便携式应用中的潜力巨大。

闪存设备称为**闪存驱动器**（flash drive），容量可达到几GB，可用于一般的海量存储应用。闪存设备被封装在小的塑料格子里（长约3英寸），其一端有一个可以取下的帽，当驱动器处于脱机状态时，可以保护这个设备的电子连接器。这些便携设备容量大，很容易连接到计算机以及从计算机断开，对于脱机状态的数据存储是很理想的选择。不过，由于它们的微小存储晶格的缺点，当涉及真正长期应用时，它们不如光学盘片可靠。

闪存技术的另一应用是**SD存储卡**（Secure Digital memory card），简称SD卡。SD卡的容量高达2 GB，它们被制成塑料封装的晶圆，有邮票大小（事实上还有更小的小型和微型SD卡），**SDHC存储卡**（Secure Digital High Capacity memory card，高容量SD存储卡）的存储容量可以高达32 GB，而作为新一代SD卡的**SDXC存储卡**（Secure Digital Extended Capacity memory card），其容量可超过1 TB。凭借不占空间的体积，这些卡可以方便地插入小型电子设备的插槽。因此，

① DVD全称也作Digital Video Disc。——编者注

它们是数码相机、智能手机、音乐播放器、汽车导航系统，以及其他许多电子应用的理想选择。

1.3.4 文件存储及检索

海量存储系统中的信息一般被分组为较大的单元，即**文件**（file）。典型的文件可能由文本文档、照片、程序、音乐录音或者一组有关公司员工的数据组成。我们已经了解到，海量存储设备规定这些文件要以较小的多字节单元进行存储和检索。例如，存储在磁盘上的文件必须按照扇区操作，每个扇区都有固定的规格。符合存储设备特性的数据块称为**物理记录**（physical record）。因此，海量存储系统中的大文件通常包含多个物理记录。

与这种物理记录划分相对，文件通常还有其自然划分，这由它所表示的信息决定。例如，一个包含公司员工信息的文件由许多单元组成，其中每个单元包含一个员工的信息；一个有关文本的文件包含段落或页。这些自然产生的数据块称为**逻辑记录**（logical record）。

逻辑记录通常由称为**字段**（field）的较小单元组成。例如，一个包含员工信息的逻辑记录大致由姓名、地址、员工标识号等字段组成。有时候，文件的每一个逻辑记录是由一个特定的字段唯一标识出来的（也许是一个员工的标识号、一个部门标号或者是目录项标号）。这样的标识字段称为**键字段**（key field），键字段中的值称为**键**（key）。

逻辑记录的规格很少能与海量存储系统的物理记录相匹配。因此，人们可能会发现若干逻辑记录存放在一个物理记录里，或者一个逻辑记录存放在两个或者更多的物理记录里（见图1-12）。因此，海量存储系统的信息检索需要一定的整理工作。这个问题的一个常用解决方法是，在主存储器里留出一个足够大的区域，用于存放若干物理记录并将此存储空间作为重组区域。也就是说，与物理记录兼容的数据块可以在主存储区与海量存储系统之间传输，主存储区的数据能够根据逻辑记录引用。

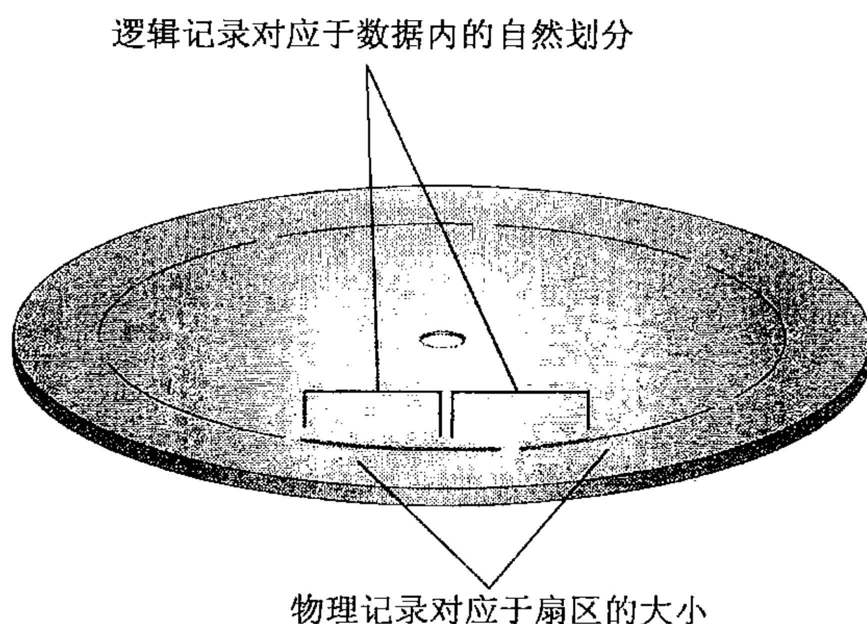


图1-12　磁盘上的逻辑记录与物理记录

这种存储区域称为**缓冲区**（buffer）。一般情况下，缓冲区是在一个设备向另一个设备传输数据的过程中临时存储数据的区域。例如，现代的打印机都有自己的存储电路，其大部分被用作缓冲区，用于保存该打印机已经收到但还没有打印的那部分文档。

问题与练习

1. 我们可以从加快磁盘或CD转速中获得什么？
2. 当记录数据到多盘片存储系统时，我们是应该写满一张盘片后再写另一张盘片，还是应该写满一个柱面后再写另一个柱面？

3. 为什么在一个预订系统里，那些需要经常更新的数据要存储在磁盘里，而不是CD或DVD里？
4. 使用字处理程序修改文档时，有时添加一段文本不会很明显地增加海量存储器中文件的大小，而有时一个符号的增加就会使文件增加几百个字节。为什么？
5. 相对于本节介绍的其他海量存储系统，闪存驱动器有什么优势？
6. 什么是缓冲区？

1.4 用位模式表示信息

在研究了位存储的技术后，现在来了解如何将信息编码为位模式。我们的学习集中在对文本、数字数据、图像以及声音等编码的流行方法上，其中每一个编码系统都可能会影响到典型的计算机用户。我们的目标是充分了解这些技术，以便知道应用这些技术的效果。

1.4.1 文本的表示

文本形式的信息通常由一种代码表示，其中文本中的每一个不同的符号（例如字母和标点符号等）均被赋予唯一的位模式。这样，文本就表示为一个长的位串，连续的位模式逐一表示原文本中的符号。

在20世纪的40年代～50年代，人们设计了许多这样的代码，并结合不同的设备使用，随之增加了不少通信问题。为了缓解这种情况，ANSI（American National Standards Institute，美国国家标准化学会）采用了ASCII（American Standard Code for Information Interchange，美国信息交换标准码）。这种代码使用长度为7的位模式来表示大小写英文字母、标点符号、数字0～9以及某些控制字符，如换行、回车与制表符等。今天，ASCII码被扩展为8位位模式，方法就是在每个7位位模式的最高端添加一个0。这个技术不仅使所产生的代码的位模式与字节型存储单元相匹配，而且还提供了附加的128个位模式（通过给附加的位赋予数值1），可以表示除英语字母和关联的标点符号之外的符号。

美国国家标准化学会

美国国家标准化学会（ANSI）成立于1918年，学会下属了一些工程师协会和政府机构，作为非赢利性组织来规范私人企业自发标准的开发。今天，ANSI有1300多个成员，其中包括商业组织、专业组织、行业协会以及政府代表。ANSI的总部设在纽约，它代表美国作为ISO的成员。它的网站是http://www.ansi.org。

其他国家类似的组织包括澳大利亚标准组织、加拿大标准委员会、中国国家质量技术监督局、德国标准学会、日本工业标准委员会、墨西哥标准指导委员会、俄罗斯联邦国家标准和度量委员会、瑞士标准化协会和英国标准学会。

ISO——国际标准化组织

国际标准化组织（常称为ISO）建立于1947年，是世界范围标准化实体联盟，这些实体分别来自各个国家。现如今，它的总部设在瑞士日内瓦，有100多个实体会员和许多观察会员。（观察会员也是一些国家的标准化实体，这些国家还没有全国统一的标准化实体。这些会员不能直接参与标准的开发，但可以了解ISO的活动。）ISO的网站是http://www.iso.org。

8位模式的一部分ASCII码可见附录A。利用这个附录，我们可以将位模式

01001000　01100101　01101100　01101100　01101111　00101110

解码为报文“Hello.”，见图1-13。

01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

图1-13　报文“Hello.”的ASCII码

ISO（International Organization for Standardization，国际标准化组织，这个组织的缩写也暗指了希腊语中意味“平等”的单词“isos”）开发了大量ASCII扩展，每种扩展都是针对某一主要语种设计的。例如，其中一个标准提供了表达大部分西欧语言文本所需的符号。在其128个附加模式中有表示英磅和德语元音ä、ö、ü的符号。

ISO扩展的ASCII标准在支持全世界多语通信方面取得了巨大进展，但是仍有两个主要障碍。首先，扩展的ASCII中额外可用的位模式数不足以容纳许多亚洲语言和一些东欧语言的字母表。其次，因为一个特定文档只能在一个选定的标准中使用符号，所以无法支持包含不同语种的语言文本的文档。实践证明，这两者都会严重妨碍其国际化使用。为弥补这一不足，Unicode在一些主要软硬件厂商的合作下诞生了，并迅速赢得了计算机行业的支持。这种代码采用唯一的16位模式来表示每一个符号。因此，Unicode由65 536个不同的位模式组成——足以表示用中文、日文和希伯来文等语言书写的文本。

由一长串根据ASCII或Unicode编码的符号组成的文件常称为**文本文件**（text file）。重要的是要区别下面两类文件：一类是由称为**文本编辑器**（text editor，常简称为编辑器）的实用程序操作的简单文本文件；一类是由**字处理程序**（word processor），如微软的Word产生的较复杂的文件。两者都是由文本材料组成的，但是，文本文件只包含文本中各个字符的编码，而由字处理程序产生的文件还包含许多专用格式码，用于表示字体变化、对齐信息等。

1.4.2　数值的表示

当所记录的信息只有数值时，以字符编码的形式存储信息效率就会很低。为了了解其中的原因，让我们来看看数值25的存储问题。如果我们坚持用ASCII编码符号来存储，每个符号一个字节，那么总共需要16个二进制位。此外，用16个二进制位可以存储的最大数是99。不过，我们马上就可以看到，使用**二进制记数法**（binary notation），16个二进制位可以存储0～65 535范围内的任何一个整数。因此，二进制记数法（或它的变体）被广泛应用于计算机存储器中数值数据的编码。

二进制记数法是一种数值表示方法，只使用数字0和1，区别于传统的使用数字0、1、2、3、4、5、6、7、8和9的十进制记数系统。我们将在1.5节中更详细地学习二进制记数法，现在只需要初步了解该系统。我们来考虑一种老式的汽车里程表，它的显示轮只包含数字0和1，而不是传统的十进制数字0～9。里程表以全0读数开始，当汽车行驶几英里时，最右方的滚动显示轮从0旋转至1；当这个1旋转回0时，就使得一个1出现在它的左边，因此产生模式10；接着右边的0旋转为1，产生11；这时，最右边的数从1旋转回0，使得它左边的1也旋转回0，这就使另一个1出现在第3位上，产生模式100。简言之，在我们驾驶汽车时将看到下列顺序的里程表读数：

0000
0001

```
0010
0011
0100
0101
0110
0111
1000
```

这个序列包括了整数0～8的二进制表示。尽管有些冗长乏味，但是我们可以扩展这种计数技术，用以发现16个1组成的位模式是可以表示数值65 535的，这就证实了我们的说法：0～65 535范围内的任何整数都可以利用16个二进制位进行编码。

由于它的高效性，数字信息通常以二进制记数法的某种形式存储，而不用符号编码。我们称其为“二进制记数法的某种形式”，这是因为上面描述的简单二进制系统只是计算机里应用的若干数值存储技术的基础。二进制系统的某些变体将在本章的后面讨论。现在我们只需要知道，称为**二进制补码**（two’s complement）记数法（见1.6节）的系统通常用于存储整数，因为它提供了一种便利地表示负数和正数的方法。为了表示$4\frac{1}{2}$和$\frac{3}{4}$这样带有分数部分的数，我们还要使用一种称为**浮点**（floating-point）记数法的方法（见1.7节）。

1.4.3 图像的表示

通常将图像表示为一组点，每一个点称为一个**像素**（pixel，是picture element的缩写），每个像素的显示被编码，整个图像就表示成这些已编码像素的集合，这个集合被称为**位图**（bit map）。这种方法很常用，因为许多显示设备（如打印机和显示器）都是在像素的概念上进行操作的。因此，位图格式的图像更便于显示。

在位图中的像素编码方式随着应用的不同而不同。对于黑白图像，每个像素由一个位表示，位的值取决于相对应像素是黑还是白。大多数的传真机采用此方法。对于更加精致的黑白照片，每个像素由一组位（通常是8个）表示，这就使得许多灰色阴影也可以表示出来。

就彩色图像而言，每个像素通过更为复杂的系统来编码。有两种方法很常用，我们称其中一种为RGB编码，每个像素表示为3种颜色成分——红、绿、蓝，它们分别对应于光线的三原色。一个字节通常用来表示每一个颜色成分的强度。因此，要表示原始图像中的一个单独像素，就需要3个字节的存储空间。

一个较常用的可以替代简单RGB编码的方法采用一个“亮度”成分和两个颜色成分。这时候，“亮度”成分（称为像素亮度）基本上就是红、绿、蓝部分的总和。（事实上，它是像素中白光的数量，但是现在我们不需要考虑这些细节。）其他两种成分（称为蓝色度和红色度）分别取决于在像素中所计算的像素亮度与蓝或红光数量之间的差。这3个成分合起来就包括了显示像素所需的信息。

利用亮度和色度成分进行图像编码这种方式的普及源自彩色电视领域，因为这种方法提供了可以同样兼容老式黑白电视接收器的彩色图像编码方式。事实上，只需要对彩色图像的亮度成分编码就可以制造出图像的灰度形式。

位图技术的一个缺陷在于，图像不能轻易调节到任意大小。基本上，增大图像的唯一途径就是变大像素，而这会使图像呈现颗粒状。（这就是应用于数码相机的“数字变焦”技术，与此相对的“光学变焦”是通过调整相机镜头实现的。）

为了避免缩放问题，表示图像时还可以把图像表示成几何结构的集合（如直线和曲线），这些几何结构可以用解析几何技术来编码。这种描述允许最终显示图像的设备决定几何结构的显示方式，而不是让设备再现特殊像素模式。这种方法被用在当今的字处理系统中，用于产生可缩放的字体。例如，TrueType（由微软和苹果开发）是用几何结构描述文本符号的系统，而PostScript（由Adobe系统开发）提供了一种描述字符及更一般的图形数据的方法。这种表示图像的几何方法也在CAD（Computer-Aided Design，计算机辅助设计）系统中很常见，用于在计算机屏幕上显示和操控三维物体的绘制。

对使用许多绘图软件（如微软的绘图工具）的用户来说，用几何结构表示图像与用位图表示图像之间的区别是明显的，这些绘图软件支持用户绘制的图中包含预先设定的形状（如矩形、椭圆形、基本线条等）。用户仅从菜单中选择所需的几何形状，然后使用鼠标绘制这个形状。在绘制过程中，软件保存了所画形状的几何描述。当鼠标给出方向后，内部的几何表示就被修改，再转化成位图形式显示出来。这种方法方便图像的缩放和形状的改变。然而，一旦绘制过程完成，就会去除基本的几何描述，仅保存位图，这意味着再做其他修改需要经历冗长的一个像素接一个像素的修改过程。另外，一些绘图系统会将描述作为几何图形保存下来并允许在之后进行修改。有了这些系统，就可以轻松地调整图形的大小，并可按各种尺寸显示清晰图像。

1.4.4 声音的表示

为了便于计算机存储和操作，对音频信息进行编码的最常用方法是，按有规律的时间间隔对声波的振幅采样，并记录所得到的数值序列。例如，序列0、1.5、2.0、1.5、2.0、3.0、4.0、3.0、0可以表示这样一种声波，即它的振幅先增大，然后经短暂的减小，再回升至较高的幅度，接着又减回至0（见图1-14）。这种技术采用每秒8000次的采样频率，已经在远程语音电话通信中使用了许多年。通信一端的语音被编码为数字值，表示每秒8000次的声音振幅。接着将这些数值通过通信线路传输到接收端，用来重现声音。

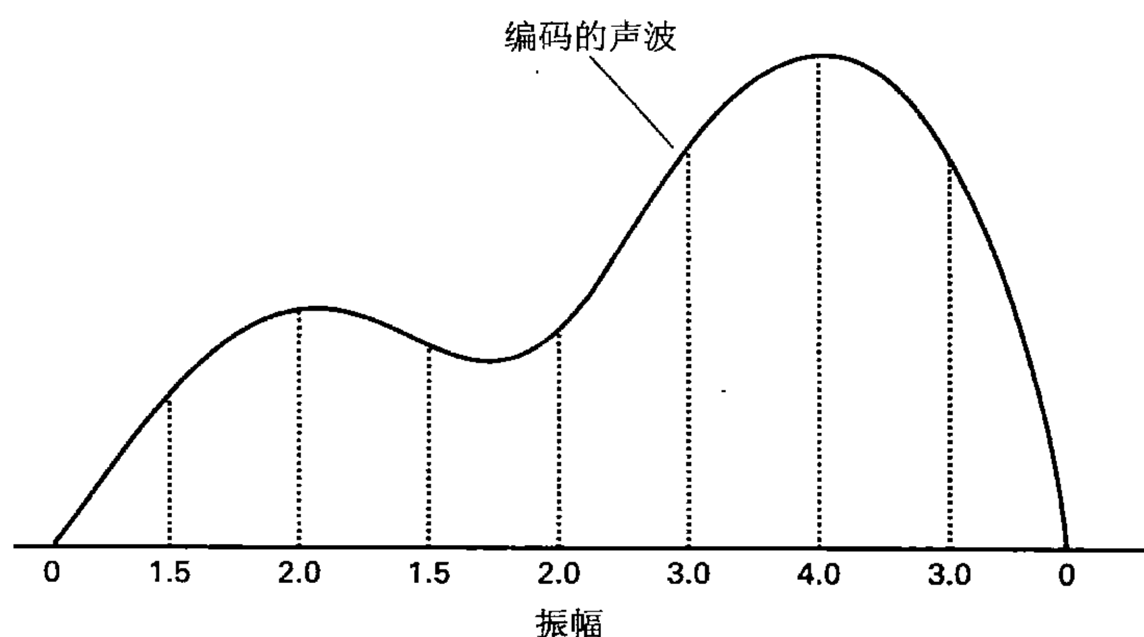


图1-14 序列0、1.5、2.0、1.5、2.0、3.0、4.0、3.0、0所表示的声波

尽管每秒8000次的采样频率似乎是很快的速率，但它还是满足不了音乐录制的高保真。为了实现今天音乐CD重现声音的质量，我们需要采用每秒44 100次的采样频率。每次采样得到的数据以16位的形式表示出来（32位用于立体声录制）。因此，录制成立体声的每一秒音乐需要100多万个存储位。

乐器数字化接口（简称MIDI）是另外一种编码系统。它广泛应用于电子键盘的音乐合成器，用来制作视频游戏的声音以及网站的辅助音效。MIDI是在合成器上编码产生音乐的指令，而不

是对音乐本身进行编码，因此它避免了采样技术那样的大存储容量要求。更精确地说，MIDI是对什么乐器演奏什么音符以及持续时间进行编码。例如，单簧管演奏D音符2秒钟，可以编码为3个字节，而不必按照每秒44 100次的采样频率用两百多万个二进制位来编码。

简言之，可以把MIDI看做是对演奏者乐谱编码的一种方法，而不是对演奏本身编码。因此，MIDI“录制”的音乐在不同合成器上演奏时声音可能是截然不同的。

问题与练习

1. 下面是ASCII编码的一条消息，每个符号8位。它的含义是什么？（见附录A。）

   01000011 01101111 01101101 01110000 01110101 01110100
   01100101 01110010 00100000 01010011 01100011 01101001
   01100101 01101110 01100011 01100101
2. 在ASCII码中，大写字母码和相应小写字母码之间的关系是什么？（见附录A。）
3. 用ASCII对这些语句编码：
   a. “Stop!” Cheryl shouted.
   b. Does 2+3＝5?
4. 描述一种在日常生活中能够呈现两种状态的设备，例如旗杆上的旗帜，或者升起或者下降。给一种状态赋值1，另一种赋为0。然后让我们看一下，当以这样的位来存储时，字母b的ASCII码会怎样表示？
5. 将下列二进制表示分别转化为相应的十进制形式。
   a. 0101　b. 1001　c. 1011
   d. 0110　e. 10000　f. 10010
6. 将下列的十进制表示分别转化为相应的二进制形式。
   a. 6　b. 13　c. 11
   d. 18　e. 27　f. 4
7. 如果每个数字采用每字节一个ASCII码的模式编码，那么3个字节可以表示的最大数字值是多少？如果采用二进制编码，那么又能够表示多大的数字值？
8. 除十六进制以外，另一种表示位模式的方法是**点分十进制记数法**（dotted decimal notation），其中每个字节由相对应的十进制数来表示，而且这些字节表示用句点分开。例如，12.5表示0000110000000101模式（12表示00001100字节，5表示00000101字节），而136.16.7表示100010000001000000000111模式。用点分十进制记数法表示下列位模式：
   a. 0000111100001111　b. 001100110000000010000000
   c. 0000101010100000
9. 相对于位图技术，用矢量技术表示图像有哪些优点？位图技术相对于矢量技术又有哪些优点？
10. 假如采用文中所讨论的每秒44 100次的采样频率，给立体声录音的1小时音乐编码。请问这段音乐编码的大小与CD的存储容量相比结果如何？

*1.5 二进制系统

在1.4节中我们看到，二进制记数法是表示数字值的一种方法，仅仅利用数字0和1；不同于普遍采用的十进制记数系统，十进制系统是利用数字0～9。现在我们要比较深入地研究一下二进制记数法。

1.5.1 二进制记数法

回顾十进制系统，每一个位置的表示都与一个量值相关联。在375的表示中，5的位置与量1

相关联，7与量10相关联，3与量100相关联（见图1-15a）。每一个量值是它右边量值的10倍。整个表达式代表的数值是，每一个数字值与其位置的量值相乘所得积之和。举例说明：模式375表示（3×100）+（7×10）+（5×1），用更加技术性的表示法即（3×10^2）+（7×10^1）+（5×10^0）。

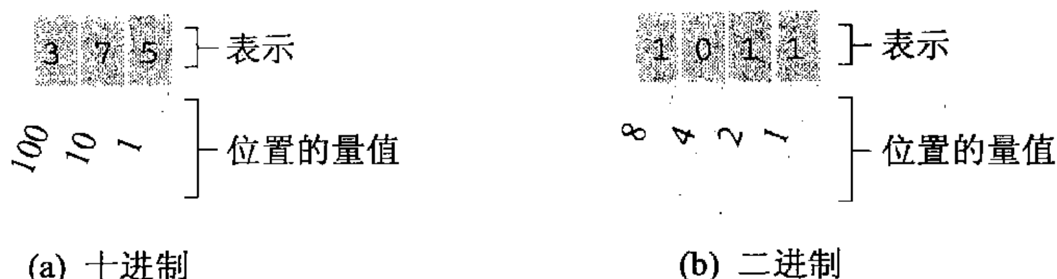


图1-15　十进制和二进制系统

在二进制记数法中，每个数字的位置也与一个量值相关联，只是与每个位置相联系的那个量值是它右边量值的两倍。更精确地说，二进制表示中最右边的数字与量值1（2^0）相关联，其左边的下一个位置与量值2（2^1）相关联，下一个与量值4（2^2）相关联，再下一个与量值8（2^3）相关联，依次类推。例如，在二进制表示1011中，最左边1的位置与量值1相关联，接下来一个1的位置与量值2相关联，0的位置与量值4相关联，最左边1的位置与量值8相关联（见图1-15b）。

为了求得二进制表示所表示的数值，我们可以采取和十进制相同的步骤，即先求得每个数字值与其量值的积，再计算各个乘积之和。例如，100101表示的数值是37，如图1-16所示。需要注意的是，因为二进制计数法仅使用数字0和1，这种求积再求和的步骤就可以简化为求数字值为1的位置对应的量值的和。因此，二进制模式1011表示的是数值11，因为3个1的位置分别与量值1、2以及8相关联。

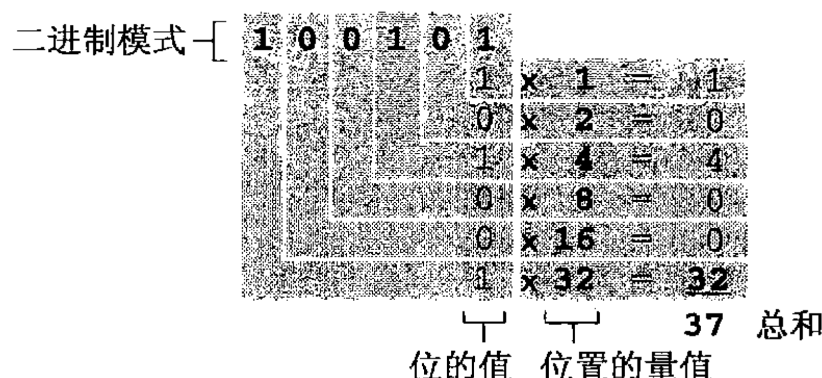


图1-16　二进制表示100101的解码

在1.4节中，已经学习了如何用二进制记数法计数，这就使得我们可以对小整数进行编码。为了求得大数值的二进制表示，你可能更倾向于图1-17所描述的算法。让我们利用这个算法来求数值13的二进制表示（见图1-18）。首先，将13除以2，得到商数6和余数1。因为这个商不是0，步骤2告诉我们还要在商数（6）的基础上除以2，得到新的商数3和余数0。最新的商数仍然不为0，所以再除以2，得出商数1和余数1。再一次，将最新的商数除以2，此时得到商数0和余数1。因为现在的商数是0，我们进入步骤3，从余数列中得到原数（13）的二进制表示1101。

步骤 1	将该值除以2，记下余数。
步骤 2	只要所得的商不是零，就继续将最新的商除以2，并记下余数。
步骤 3	商为0时，将余数按所记录的顺序从右到左依次排列，即得到原数的二进制表示。

图1-17　求正整数二进制表示的算法

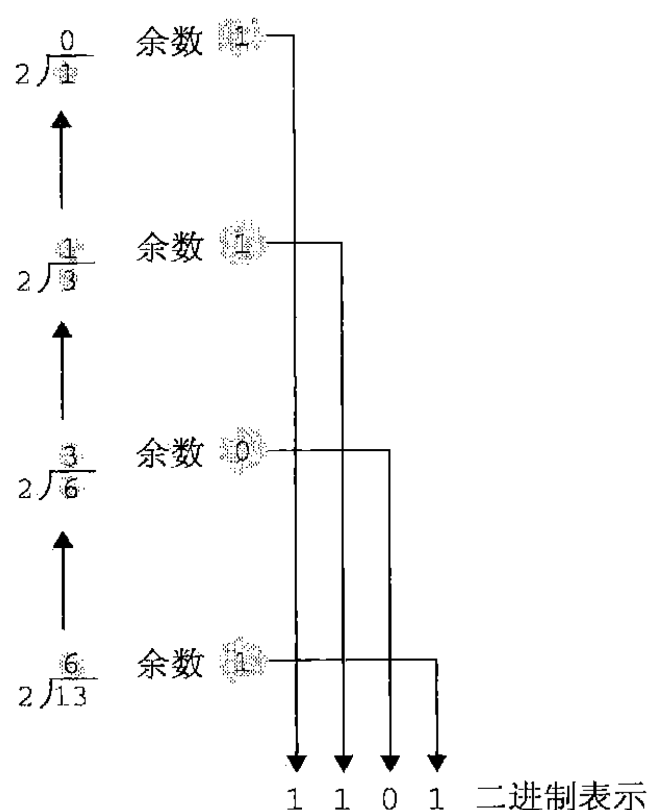


图1-18 利用图1-17的算法求13的二进制表示

1.5.2 二进制加法

为了理解两个用二进制表示的整数的相加过程，首先让我们回顾一下用传统十进制表示的数值的相加过程。例如，考虑下面的问题：

```
  58
+ 27
```

我们先对最右列的8和7相加，得到和为15，我们把5记录在这一列的底部，进位1放到下一列中，得到：

```
   1
  58
+ 27
   5
```

现在我们把下一列的5和2相加，并加上进位到这一列的1，得到的和为8，我们把8记录在这一列的底部，得到：

```
  58
+ 27
  85
```

总之，这个过程就是从右到左相加每一列中的数字，把和中的零头数字写在列的底部，把和的大数（如果有）进到下一列。

为了相加两个用二进制表示的正整数，我们遵照相同的过程，只是所有和的计算都使用图1-19中显示的加法规则，而不是你在小学所学的传统的以10为基的加法规则。例如，为了解决问题：

```
  111010
+  11011
```

首先相加最右边的0和1，得到1，写于该列下方。接着相加下一列的1和1，得到10。把其中的0写于该列下，并将1记在了下一列的上面。这时，加法如下：

```
      1
  111010
+  11011
      01
```

相加下一列的1、0和0，得到1，将1写于该列下。下一列的1和1总和为10，将0写于该列下，并将1记于下一列。这时，加法如下：

```
    1
  111010
+  11011
    0101
```

下一列的1、1和1总和为11（数值3的二进制符号），将低位1写于该列下方，并将另外一个1写在了下一列的上面。把那个1与那列原本的1相加，得到10。再一次，在该列下方写下低位0，并将1写在了下一列上面。现在得到

```
 1
  111010
+  11011
  010101
```

下一列的唯一项就是1，是上一列进过来的，所以我们将其记录为答案。最终的结果是：

```
  111010
+  11011
 1010101
```

```
  0     1     0     1
 +0    +0    +1    +1
  0     1     1    10
```

图1-19　二进制加法法则

1.5.3　二进制中的小数

为了扩展二进制记数法，使其包含小数数值，我们使用了**小数点**（radix point），其功能与十进制符号中的十进制小数点是相同的。也就是说，小数点左边的数字代表整数部分（整个部分）的数值，如同前面讨论的二进制系统那样解释，而小数点右边的数字代表数值的小数部分，解释类似其他二进制位，只是它们的位置被赋予了小数的量值。也就是说，小数点右边第一位的量值是1/2（2^{-1}），下一位的量值是1/4（2^{-2}），再下一位是1/8（2^{-3}），依次类推。需要注意的是，这仅仅是前面所述规则的延续，即每位所被赋予的量值是它右边大小的两倍。利用这些赋予二进制位位置的量值，对包含小数点和不包括小数点的二进制表示进行解码的步骤基本是相同的。更精确地说，我们把表示中每一个位值与其对应位位置的量值相乘。举例说明，二进制记数法表示的101.101，将其解码可得$5\frac{5}{8}$，见图1-20。

图1-20　二进制表示101.101的解码

应用于十进制系统里的加法技术同样适用于二进制系统。也就是说，对两个有小数点的二

进制表示的数相加，我们只需要对齐小数点，然后像从前一样应用相同的加法步骤。例如，10.011加100.11得111.001，如下所示：

```
   10.011
+ 100.110
  111.001
```

问题与练习

1. 将下列每个二进制表示转换为相应的十进制形式。
   a. 101010　b. 100001　c. 10111　d. 0110　e. 11111
2. 将下列每个十进制表示转换为相应的二进制形式。
   a. 32　b. 64　c. 96　d. 15　e. 27
3. 将下列每个二进制表示转换为相应的十进制形式。
   a. 11.01　b. 101.111　c. 10.1　d. 110.011　e. 0.101
4. 用二进制记数法表示下列数值。
   a. $4\frac{1}{2}$　b. $2\frac{3}{4}$　c. $1\frac{1}{8}$　d. $\frac{5}{16}$　e. $5\frac{5}{8}$
5. 按照二进制记数法做下列加法。

```
a.   11011    b.  1010.001    c.   11111    d.   111.11
   + 1100       +    1.101       + 0001        + 00.01
```

*1.6 整数存储

数学家们长久以来就对数字记数系统很感兴趣，而且他们的许多想法已经被证明与数字电路的设计是相符的。本节，我们将研究其中两种记数系统，二进制补码记数法和余码记数法。它们都用于在计算设备中表示整数。这些系统都是基于二进制系统的，但是增加了些其他的特性，因而与计算机设计更加匹配。尽管有这么多的优点，它们还是有缺陷的。我们的目标是了解这些特性以及它们是如何影响计算机用法的。

模拟与数字

在21世纪之前，许多研究人员都在讨论数字和模拟技术的优缺点。在一个数字系统里，数值被编码成一系列数字并存储在若干存储单元中，每个单元表示一个数字。在一个模拟系统里，每个数值存储在单独的一个存储单元里，它在一个连续的范围内可以表示任何数值。

让我们利用水桶作为存储器来比较这两种方法。为了模拟一个数字系统，我们让一个空桶表示数字0，一个满桶表示数字1，然后我们就可以利用浮点记数法（见1.7节）用一排水桶存储一个数字值。相反，为了表示模拟系统，我们用水桶水位表示数字值。乍一看，模拟系统看起来更精确，因为它不会有数字系统中的截断误差（见1.7节）。不过，在模拟系统中水桶的任何移动都会使水位检测出错，而在数字系统中没有剧烈的晃动是不会区分不出水桶有水没水的。因此，数字系统不像模拟系统对错误那样敏感。由于数字系统的这种健壮性，许多原来基于模拟技术的应用（例如电话通信、视频录制和电视）都转而使用数字技术。

1.6.1 二进制补码记数法

今天计算机表示整数最普遍的系统就是**二进制补码**（two’s complement）记数法。这个系统

采用固定数目的二进制位来表示系统中的每一个数值。在今天的设备中，应用二进制补码系统是很普遍的，其中每个数值用一个32位的模式表示。这种大系统方便表示很大范围的数字，但对于教学不是很便利。因此，学习二进制补码系统的特性时，我们将把精力集中在比较小的系统上。

图1-21列出了两种二进制补码系统——一种基于长度为3的位模式，另一种基于长度为4的位模式。这种系统是这样构成的，即先规定适当长度的一组二进制0，接着用二进制计数，直到只有一个0，其他都是1的模式形成。这些模式表示数值0, 1, 2, 3,…。表示负值的模式是这样获得的，即先规定一组适当长度的二进制1，接着按照二进制反向计数，直到只有一个1，其他都是0的模式形成。这些模式表示数值−1, −2, −3,…。（如果你认为利用二进制反向计数有困难，那么可以仅从表格底部，即只有一个1，其他都为0的模式开始，计数到全是1的模式。）

位模式	所表示的值
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

(a) 使用长度为3的位模式

位模式	所表示的值
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

(b) 使用长度为4的位模式

图1-21 二进制补码记数法系统

注意，在二进制补码系统中位模式最左边的二进制位指明所表示数值的符号。因此，最左边的位常称为**符号位**（sign bit）。在二进制补码系统中，符号位为1的模式表示负值，符号位为0的模式表示非负值。

在二进制补码系统中，绝对值相同的正负数值之间的模式很相近，从右向左读时，直到第一个二进制1，它们都是相同的。然后，以这个1为分界线，左面的位模式互为补码。（一个模式的**补码**（complement）是通过转换所有的二进制0为1，并转换所有的二进制1为0得到的模式。）例如，图1-21中的4位系统，表示2和−2的模式都是以10结束，但是表示2的模式开始为00，而表示−2的模式开始为11。观察到这一点，我们就可以得出在绝对值相同的、表示正负值的位模式之间转换的算法。我们只需要从右到左复制原始的模式直到第一个1，接着在将剩余位转换为最终位模式时，对这些剩余位取反（图1-22）。

理解了二进制补码系统的这些基本特性，也可以得出一个二进制补码表示法的解码算法。如果要解码的模式有一个符号位0，我们仅仅需要读出这个数值，就好像这个模式是一个二进制表示。例如，0110表示数值6，因为110是6的二进制表示。如果要解码的模式有一个符号位1，就知道表示的数值是负的，而我们所要做的就是找到其绝对值。为了实现这个目的，我们先要利用图1-22中“复制及取反”的步骤，然后对获得的模式进行解码，就仿佛它只是一个简单的二进制表示。例如，为了对模式1010解码，首先我们意识到，因为这个符号位是1，表示的数值

就是负的。因此，我们利用“复制及取反”步骤获得了模式0110，认识到这是6的二进制表示，然后得出结论：原始的模式表示−6。

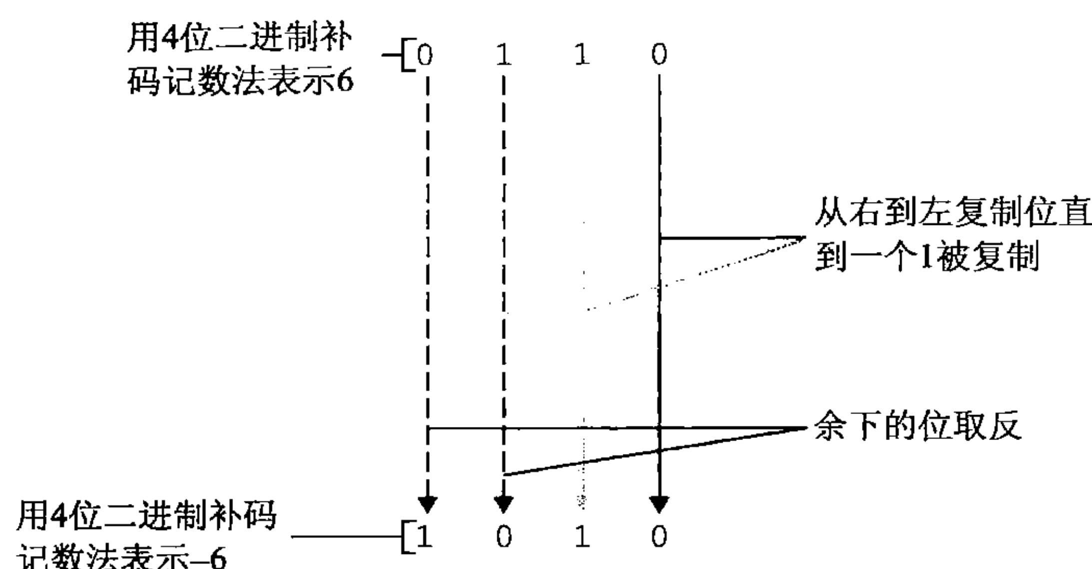


图1-22 利用二进制补码记数法用4个位对数值6编码

1. 二进制补码记数法中的加法

我们采用二进制加法中使用的算法来计算二进制补码记数法中的数值相加，只是包括答案的所有位模式长度都相同。这就意味着，在二进制补码系统的加法中，由于最后一个进位，答案左边产生的任何一个附加位都要删除。因此，“加法运算”0101和0010得出0111，0111和1011得出0010（0111+1011=10010，缩减为0010）。

根据这个理解，我们来分析一下图1-23中的3个加法问题。每一个情况，我们都把问题转化为二进制补码记数法（采用长度为4的位模式），演示先前描述过的加法过程，然后对结果进行解码，回到一般的十进制记数法。

十进制问题	二进制补码问题	十进制答案
3 + 2	0011 + 0010 = 0101	5
−3 + −2	1101 + 1110 = 1011	−5
7 + −5	0111 + 1011 = 0010	2

图1-23 转换为二进制补码记数法的加法问题

注意，图1-23的第3个问题涉及正值和负值的加法，它展示了二进制补码记数法的一个主要优点：任何带符号数字组合的加法都可以利用相同的算法，于是也就可以用相同的电路。这与人们传统的计算法是截然相反的。尽管小学生先学加法，然后是减法，但是应用二进制补码记数法的计算机只需知道加法就可以了。

例如，减法问题7−5与加法问题7+（−5）是一样的。因此，如果人们命令计算机执行7（存储为0111）减5（存储为0101），那么它首先要转换5为−5（表示为1011），然后执行0111+1011的加法过程，得到代表数值2的0010，如下所示：

```
 7          0111          0111
-5    →   - 0101    →   + 1011
                          0010   →   2
```

因此我们可以看到，当二进制补码记数法用于表示数字值时，一个加法电路与一个取负电

路的组合就足以解决加法以及减法的问题了。（这些电路的图示及解释详见附录B。）

2. 溢出问题

我们在前面的例子中忽略了这样一个问题，就是在任意的一个二进制补码系统中，都有对所表示数值大小的限制。当使用4位模式二进制补码时，可以表示的最大正整数是7，最小负整数是−8。具体来说，数值9无法被表示出来，这就意味着我们不能指望得出5+4的正确答案。事实上，它的结果会为−7。这种现象称为**溢出**（overflow）。也就是说，溢出指的是这样一个问题，即计算得出的数值超出了可以表示的数值范围。使用二进制补码记数法时，两个正值或负值分别相加都可能会出现这种情况。无论哪种情况，检查答案的符号位就可以发现溢出的条件。如果两个正值相加的结果是负值的模式，或者两个负值相加的结果为正，那么就发生了溢出问题。

当然，使用二进制补码系统的大多数计算机的位模式都比例子中的长，因而在进行较大数值操作时不会产生溢出。今天，人们普遍使用二进制补码记数法的32位模式来存储数值，可以得到的最大正值是2 147 483 647。如果需要更大的数值，我们可以使用更长的位模式，或者改变度量单位。例如，若在解答一个问题时用英尺代替英寸，所得数值就变小了，而且也可以达到所要求的精确度。

关键问题是计算机会制造错误。因此，使用计算机的人一定要意识到可能涉及的危险。其中一个问题就是，计算机程序员和使用者会自满而导致忽视一个事实，即小数值可以累加成大数值。例如，人们过去普遍使用二进制补码记数法的16位模式表示数值，这就意味着出现大于或等于2^{15}=32 768的数值时就会产生溢出。1989年9月19日，一家医院多年来运行良好的计算机出现了故障。仔细检查后发现，那天距1900年1月1日共32 768天，而计算机的程序正是基于那个起始日期开始计算日期的。因此，由于溢出原因，1989年9月19日的日期产生了负值——设计计算机程序时没有考虑到这种现象。

1.6.2　余码记数法

表示整数值的另外一种方法是**余码记数法**（excess notation）。与二进制补码记数法相同，余码记数法中的每一个数值都表示为相同长度的位模式。为了建立一个余码系统，我们首先选择所使用的模式的长度，然后根据二进制记数呈现的顺序写下那个长度的所有位模式。接着我们发现，二进制1作为其最高位的第一个模式大约就在数列的中间。我们用这个模式表示0，其前的模式就分别用于表示−1, −2, −3,…，其后的模式分别用于表示1, 2, 3,…。使用长度为4的模式产生的编码见图1-24。我们可以看到，模式1101表示数值5，0011表示数值−5。（注意，余码系统和二进制补码系统的区别就是符号位相反。）

位模式	所表示的值
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

图1-24　余8代码转换表

图1-24表示的系统称为余8记数法。为了了解其由来，首先我们用传统二进制系统的编码翻译每一个模式，然后将其与余码记数法表示的数值进行比较。对于每一个模式，你会发现二进制解释值比余码记数法解释值都要大8。例如，模式1100用二进制记数法表示为数值12，在余码系统中则表示4；0000用二进制记数法表示为数值0，但是在余码系统中则表示为−8。与此类似，在基于长度为5的位模式的余码系统中，模式10000用于表示0而不是通常的数值16，该记数法称为余16记数法。同样，你可以证明3位余码系统应该称为余4记数法（图1-25）。

位模式	所表示的值
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

图1-25　使用长度为3的位模式的余码记数系统

问题与练习

1. 将下面每一个二进制补码表示转换为相应的十进制形式。

   a. 00011　b. 01111　c. 11100　d. 11010　e. 00000　f. 10000

2. 用8位位模式将下列每一个十进制表示转换为相应的二进制补码形式。

   a. 6　b. -6　c. -17　d. 13　e. -1　f. 0

3. 假定下列位模式表示的是用二进制补码记数法存储的数值，求出每一个值的负值的二进制补码表示。

   a. 00000001　b. 01010101　c. 11111100

   d. 11111110　e. 00000000　f. 01111111

4. 假定一台机器用二进制补码记数法存储数值，如果机器分别采用下列长度的位模式，那么可以存储的最大数和最小数分别是什么？

   a. 4　b. 6　c. 8

5. 在下列问题中，每个位模式表示一个用二进制补码存储的数值。请执行文中所述的加法过程，按照二进制补码记数法求出它们的答案。并将问题及答案转换为十进制记数法进行验证。

   a. 0101 + 0010　b. 0011 + 0001　c. 0101 + 1010　d. 1110 + 0011　e. 1010 + 1110

6. 计算下列由二进制补码记数法表示的问题，但这次要观察溢出问题，并指出哪个答案因产生溢出而不正确？

   a. 0100 + 0011　b. 0101 + 0110　c. 1010 + 1010　d. 1010 + 0111　e. 0111 + 0001

7. 将下列问题从十进制记数法转换为长度为4的位模式的二进制补码记数法，然后将每一个问题转换成一个相应的加法问题（如计算机的做法），然后执行加法。将求得的答案转换为十进制记数法以进行验证。

   a. 6 − (−1)　b. 3 − 2　c. 4 − 6　d. 2 − (−4)　e. 1 − 5

8. 在二进制补码记数法里，一个正数和一个负数相加时会产生溢出吗？请说明理由。

9. 将下面每一个余8码表示转换为相应的十进制形式（解题时不要看文中的表格）。

   a. 1110　b. 0111　c. 1000　d. 0010　e. 0000　f. 1001

10. 将下列的每一个十进制表示转换为相应的余8码形式（解题时不要看文中的表格）。

    a. 5　b. -5　c. 3　d. 0　e. 7　f. -8

11. 数值9可以用余8记数法表示吗？用余4记数法表示6呢？请说明理由。

*1.7 小数的存储

不同于整数存储，对于包括小数部分的数值，我们不仅要存储代表其二进制表示的0和1，还有其小数点的位置。一种流行的方法是基于科学记数法的，称为**浮点**（floating-point）记数法。

1.7.1 浮点记数法

让我们以只用一个字节存储的例子来解释浮点记数法。尽管计算机通常使用更长的模式，这种8位格式也可以表示实际的系统，既可以表示重要的概念，又避免了长字节的混乱。

首先我们要规定这个字节的高位端为符号位。再次说明，符号位中的二进制0代表存储的数值为非负，1代表数值为负。接着，我们将这个字节剩余的7个位分为2组，或称其为域：**指数域**（exponent field）和**尾数域**（mantissa field）。我们规定符号位右边的3个位为指数域，余下的4个

位为尾数域。图1-26描述了如何拆分字节。

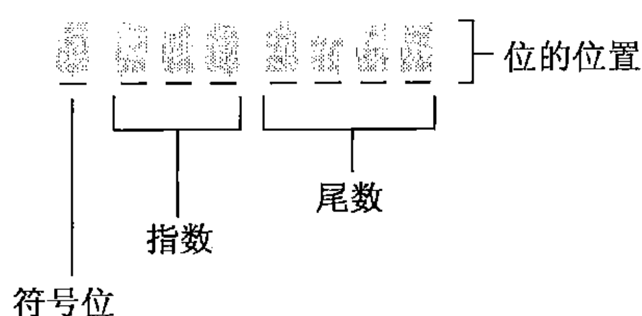


图1-26　浮点记数法成分

我们可以借助下面的例子解释这些域的含义。假如一个字节由位模式01101011组成。利用前面的形式分析这个模式，可以看出符号位是0，指数是110，尾数是1011。为了对这个字节解码，我们首先要求解它的尾数，并在它的左边放置一个小数点，于是得到

```
.1011
```

接着，我们求解指数域（110）的内容，并将其解释为一个用3位余码方法（见图1-25）存储的整数。因此，我们所举例子的指数域模式表示正数2。这就要求我们将上面所得结果的小数点向右移动2位。（负指数域就意味着向左移动小数点。）因此，我们可以得到

```
10.11
```

这就是 $2\frac{3}{4}$ 的二进制表示。接着，我们看到例子中的符号位是0，因此表示的数值是非负。可以得出结论：字节01101011表示 $2\frac{3}{4}$ 。如果模式是11101011（除了符号位都与之前相同），表示的数值就将为 $-2\frac{3}{4}$ 。

再看一个例子，字节00111100。求尾数后得到

```
.1100
```

然后将小数点向左移动一位。指数域（011）表示数值-1，因此得到

```
.01100
```

这表示3/8。因为原始模式中的符号位是0，所以存储的数值是非负。我们得出结论：模式00111100表示3/8。

用浮点记数法存储数值，我们要颠倒前面的过程。例如，为了对 $1\frac{1}{8}$ 编码，我们首先要将其用二进制记数法表示，得到1.001。接着，我们要从左到右将其位模式复制到尾数域，要从二进制表示的最左边的1开始。此时，这个字节如下：

$_\ _\ _\ _\ \underline{1}\ \underline{0}\ \underline{0}\ \underline{1}$

我们现在必须填充指数域。为了达到这个目的，假定尾数域的左边有一个小数点，然后规定位的数量以及小数点移动的方向，以此得到原始的二进制数字。我们在例子中可以看到，.1001中的小数点要向右移动一位才能得到1.001，指数因此为正，所以我们将101（在余4记数法中表示为正1，见图1-25）置于指数域。最后，因为存储的数值是非负的，我们用0填充符号位。完成的字节如下：

$\underline{0}\ \underline{1}\ \underline{0}\ \underline{1}\ \underline{1}\ \underline{0}\ \underline{0}\ \underline{1}$

当填充尾数域时，你可能会漏掉一个微妙的细节，这个规则是从左至右复制以二进制表示的位模式，并要从最左边的1开始。为阐述清楚，让我们考虑一下存储数值 $\frac{3}{8}$ 的过程，它用二进制记数法表示为.011。这时，其尾数为

$_\ _\ _\ _\ \underline{1}\ \underline{1}\ \underline{0}\ \underline{0}$

而不会是

_ _ _ _ 0 1 1 0

这是因为我们是从二进制表示最左边的1开始填充尾数域。遵循这个规则的表示称为**规范化形式**（normalized form）。

使用规范化形式减少了同一数值多种表示的可能性。例如，00111100和01000110都可以解码成$\frac{3}{8}$，但是只有第一个模式才是规范化形式。遵循规范化形式也意味着，所有非0数值的表示都会有一个以1开始的尾数。不过，数值0是一个特例，它的浮点表示就是全部为0的位模式。

1.7.2 截断误差

如果要利用1字节浮点记数法存储数值$2\frac{5}{8}$，那么让我们考虑因此会出现的恼人问题。我们首先用二进制写$2\frac{5}{8}$，得到10.101。但是，当把这个模式复制到尾数域时，我们就用尽了空间，最右边的1（表示最后的$\frac{1}{8}$）因此丢失了（图1-27）。如果现在忽视这个问题，继续填充指数域和符号位，那么我们最后得到的位模式将为01101010，它表示的是$2\frac{1}{2}$，而不是$2\frac{5}{8}$。这个现象称为**截断误差**（truncation error）或**舍入误差**（round-off error）。这就意味着，由于尾数域空间不够大，存储的部分数值丢失了。

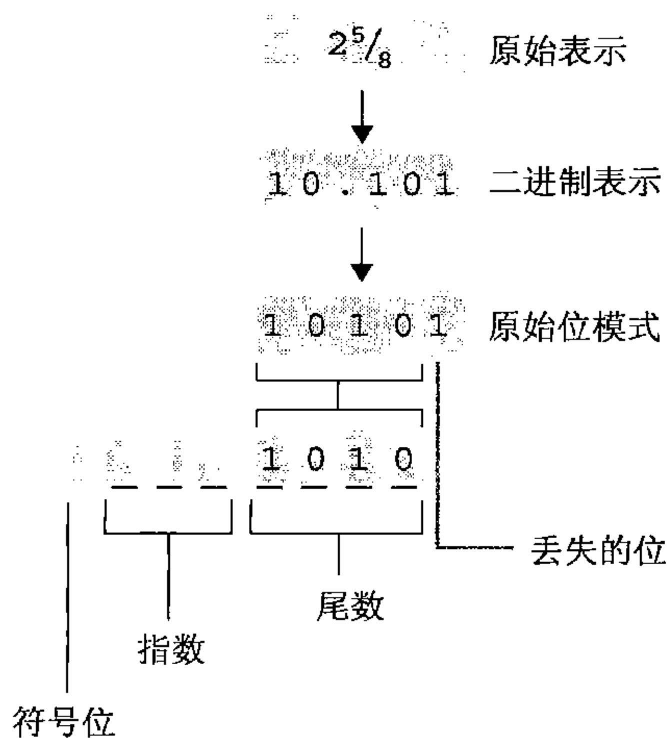


图1-27 数值$2\frac{5}{8}$的编码过程

使用较长的尾数域可以减少这种误差的发生。事实上，今天生产的大多数计算机都至少采用32位存储浮点记数法表示的数值，而不是我们在本书中采用的8位。这同时使得指数域也更长。不过，即使有这样较长的格式，有时候还是需要更高的精确度。

截断误差的另外一个来源就是在十进制记数法中比较常见的一个现象，即无穷展开式问题，例如发生在我们用十进制形式表示1/3的时候。无论我们用多少位数字，有一些数值都不能被精确地表示出来。传统的十进制记数法与二进制记数法区别在于，二进制记数法中有无穷展开式的数值多于十进制。例如，数值1/10表示为二进制时为无穷展开式。想象一下，一个粗心的人用浮点记数法存储和处理美元与美分时会产生什么样的问题？尤其是，如果美元被用作度量单

位，那么一角就不能被精确地存储。其中一个解决方式就是，以分为单位处理数据，这样所有的数值就都是整数，都可以用诸如二进制补码这样的方法精确存储。

单精度浮点数

1.7节介绍的浮点表示法过于简单，不能用于实际的计算机中。毕竟，在全部实际数字中，这一表示法的8位只能表示其中256个数。我们在讨论中使用了8位模式来保持示例的简单性，但依然涵盖了重要的基本概念。

今天的许多计算机都支持32位形式的单精度浮点表示法。这一格式使用1位表示符号位，用8位表示指数（使用超码表示法），用23位表示尾数。因此，单精度浮点最多有7位十进制有效数字，可以表示极大的数（数量级为10^{38}）直至极小的数（数量级为10^{-37}）。也就是说，给定一个十进制数，可以非常精确地存储7位十进制有效数字，但仍有可能存在少量误差）。前7位之后的数字一定会因截断误差丢失，但数字的近似值会被保留下来。另一种形式是64位的双精度浮点数，最多有15位有效数字。

截断误差和与之相关的问题是工作在数值分析领域的人们每天都很关注的问题。这个数学分支研究的是执行大规模、高精度有效计算所涉及的问题。

下面的例子可以激起任何一位数值分析家的兴趣。假设我们要应用前面定义的1字节浮点记数法来做这3个数值的加法：

$$2\frac{1}{2}+\frac{1}{8}+\frac{1}{8}$$

如果我们按照上述顺序加数值，首先就是$2\frac{1}{2}$加上$\frac{1}{8}$，得到$2\frac{5}{8}$，二进制表示为10.101。遗憾的是，因为这个数值不能被精确地存储（如同前面所看到的），我们第一步的结果最后被存储为$2\frac{1}{2}$（与其中一个加数相同）。下一步是把这个结果再加到最后的$\frac{1}{8}$上。截断误差在这里再一次出现了，最后的结果是错误的$2\frac{1}{2}$。

现在让我们以相反的顺序来加这些数值：首先将$\frac{1}{8}$加到$\frac{1}{8}$，得到$\frac{1}{4}$，其二进制表示为.01。因此，第一步的结果在一个字节里被存储为00111000，这是精确的。然后，我们将这个$\frac{1}{4}$加到数列中的下一个数值$2\frac{1}{2}$，得到$2\frac{3}{4}$，我们可以将其在一个字节里精确地存储为01101011。这次的答案是正确的。

总而言之，在浮点记数法表示的数字值加法中，它们相加的顺序很重要。问题是，如果一个大数字加上一个小数字，那么小数字就可能被截断。因此，多个数值相加的一般规则是先相加小数字，这是为了将它们累计成一个大数字（通过加到更大的数值上）。这就是前面例子中反映的现象。

今天商用软件包的设计师们在这方面做得很好，他们使没有经过培训的使用者也能很好地避免这种问题的发生。在一个典型的电子制表软件系统中，除非相加的各个数值大小差别达到10^{16}或更多，否则所得结果都是正确的。因此，如果你认为有必要对数值

```
10,000,000,000,000,000
```

加1，那么你会得到答案：

    10,000,000,000,000,000

而不是：

    10,000,000, 000,000,001

这样的问题在一些应用中是很严重的（例如导航系统），小误差可能在加法运算中累加，最终产生严重的后果。但是，对于一般的PC使用者，大多数商用软件提供的精确度已经足够了。

问题与练习

1. 用文中所述的浮点格式对下列位模式进行解码。
   a. 01001010　b. 01101101　c. 00111001　d. 11011100　e. 10101011
2. 将下列数值编码成文中所述的浮点格式。指出截断误差的出现情况。
   a. $2\frac{3}{4}$　b. $5\frac{1}{4}$　c. $\frac{3}{4}$　d. $-3\frac{1}{2}$　e. $-4\frac{3}{8}$
3. 根据文中所述的浮点格式，模式01001001和00111101中哪一个表示的值更大？描述一种确定哪个模式表示的值更大的简单过程。
4. 使用文中所述的浮点格式时，可以表示的最大值是什么？可以表示的最小正值是什么？

*1.8 数据压缩

为了存储和传输数据，在保留原有内容的条件下，缩小所涉及数据的大小是有益的（有时也是必须的）。完成这一过程的技术称为**数据压缩**（data compression）。本节，我们首先要学习普通的数据压缩方法，然后了解一些为特殊应用设计的方法。

1.8.1 通用的数据压缩技术

数据压缩方案有两类。一类是**无损**（lossless）的，另一类是**有损**（lossy）的。无损方案在压缩过程中是不丢失信息的，有损方案在压缩过程中会发生信息丢失。通常有损技术比无损技术提供更大的压缩，因此在可以忽略小错误的数据压缩中应用很广，如图像和音频压缩。

对于被压缩数据由一长串相同的数值组成的情况，普遍使用称为**行程长度编码**（run-length encoding）的压缩技术，这是一种无损方法。它的过程是，将一组相同的数据成分替换成一个编码，指出重复的成分以及其在序列中出现的次数。例如，指出一个位模式包括253个1，接着118个0，接着87个1，这要比实际列出458个位节省空间。

另外一个无损数据压缩技术是**频率相关编码**（frequency-dependent encoding），在这个系统里，用于表示数据项的位模式长度与这个项的使用频率是反相关的。这些编码是变长编码的例子，意思是项由不同长度的模式表示，而不是像Unicode等编码那样，所有符号都由16个位表示。戴维·赫夫曼的功劳是发现了一般用于开发频率相关编码的算法，人们一般称用这种方法开发的编码为**赫夫曼编码**（Huffman code）。因此，今天使用的大多数频率相关编码都是赫夫曼编码。

让我们看一个频率相关编码的例子，考虑一下编码英文文本的任务。在英文中，字母e、t、a和i的使用频率要大于字母z、q和x。因此，当为英文文本编码时，如果用短位模式表示前面的字母，用长位模式表示后面的字母，那么就会节省空间。结果得到这样一个编码，其对英文文

本的表示要比用统一长度编码时短。

在某些情况下，压缩的数据流由各个数据单元组成，每一个数据单元与其前面一个差别很小。动画的连续帧就是一个例子。这时，使用**相对编码**（relative encoding），也称为**差分编码**（differential encoding）的技术是很有用的。这些技术记录下了连续数据单元之间的区别，而不是全部单元；也就是说，每个单元是根据其与前一个单元的关系被编码的。相对编码用无损形式和有损形式都可以完成，这取决于连续数据单元之间的差别是精确地编码还是近似地编码。

还有其他流行的基于**字典编码**（dictionary encoding）技术的压缩系统。这里的术语**字典**（dictionary）指的是一组构造块，压缩的信息通过它们建造起来，而信息本身被编码成字典的一系列参照符。我们一般认为字典编码系统是无损系统，不过在学习图像压缩时我们将看到，有时候字典条目仅仅是正确数据成分的近似值，这就使其成了有损压缩系统。

字处理系统可以使用字典编码压缩文本文件，因为为了拼写检查，一些字典已经包含在这些字处理系统中，它们是很出色的压缩字典。特别值得一提的是，一个完整的单词可以编码成字典的一个单独参考符，而不是像使用ASCII和Unicode系统那样编码成一列单独的符号。字处理系统中的一个普通字典要包括大概25 000个条目，这就意味着一个条目可以用0到24 999的整数识别。这就是说，字典中一个特定条目用15位的模式就足可识别。相反，如果用到的单词包括6个字母，则它的各个符号编码在8位ASCII码中需要48位，在Unicode中需要96位。

字典编码的一个变体是**自适应字典编码**（adaptive dictionary encoding，也称为动态字典编码）。在自适应字典编码系统中，编码过程中字典是可以改变的。一个流行的例子是**LZW编码**（Lempel-Ziv-Welsh encoding，根据它的创造者Abraham Lempel、Jacob Ziv和Terry Welsh的姓氏命名）。用LZW对信息编码，人们首先用包含基础构造块的字典，信息就是用那些构造块建起来的。但是，随着人们在信息中发现更大单元，它们就被加到了字典上——意思是，这些单元未来的出现可以被编码为一个（而不是多个）字典参照符。例如，当对英文文本编码时，人们首先要用这样的字典，它要包含单独字符、数字和标点符号。但是，当信息中的单词被确认后，可以将它们加到字典中。因此，随着对信息的编码字典会逐步扩展，而随着字典的扩展，信息中更多的单词（或者是重复的单词模式）就可以编码为字典的一个参照符。

结果是，信息用一部相当大的、完全针对本信息的字典编码。但是对这条信息解码并不一定需要这个大字典，只需要原始的小字典。的确，解码过程可以与编码过程用同一个小字典。接着，随着解码进程的继续，会遇到编码过程中发现的相同单元，因此可以将它们加到字典中，作为未来编码过程的参照符。

举例说明，考虑用LZW对信息编码：

xyx xyx xyx xyx

首先用一个有3个条目的字典，第一个是*x*，第二个是*y*，第三个是空格。我们先将*xyx*编码为121，意思是这个信息的第一个模式包括第一个字典条目，接着是第二个，然后又是第一个。接着空格被编码为1213。但是因为有了一个空格，我们知道前面的字符串已经形成了一个单词，所以我们将模式*xyx*加到字典里作为第四个条目。依此类推，整个信息就被编码为121343434。

如果我们现在要对这条信息解码，用原始的3条目字典，我们将首先将起始的1213串解码为*xyx*，接下来是空格。这时我们意识到*xyx*串形成了一个单词，因此将其加到字典中作为第四个条目，同编码过程中所做的一样。我们接着对这个信息解码，发现信息中的4指的是这第四个新条目，将其解码为单词*xyx*，因此产生模式：

xyx xyx

按这种方法，我们最终把121343434串解码为：

$xyx \quad xyx \quad xyx \quad xyx$

这就是原始信息。

1.8.2 图像压缩

在1.4节中，我们已经了解到如何用位图技术对图像编码。不过，得到的位图通常是非常大的。因此，人们已经为了图像表示专门开发出许多压缩方案。

一种称为GIF（是Graphic Interchange Format的缩写，一些人读作“Giff”，还有一些人读作“Jiff”）的系统是一个字典编码系统，由CompuServe公司研制开发。它处理压缩问题的方法是，将赋予一个像素颜色的数量减少到只有256个。这些颜色的每一个红–绿–蓝组合都用3个字节编码，这256个编码被存储在一个称为调色板的表格（一个字典）里。图像中的每个像素都可以用一个字节表示，它的数值指出在256个调色条目中哪一个表示像素的颜色。（回顾：一个字节能够包括256个不同位模式中的任意一个。）需要注意的是，GIF用于任意图像时都是有损压缩系统，因为调色板中的颜色不可能与原始图像的颜色一致。

通过LZW技术将这个简单的字典系统扩展为自适应字典系统，GIF可以进一步压缩。尤其是，因为在编码过程中遇到像素模式时会将其加到字典中，所以将来遇到这些模式时就可以更加高效地编码了。因此，最终的字典是由原始调色板和一组像素模式构成的。

GIF调色板中某一个颜色通常被赋予值“透明”，意思是背景色可以透过被赋予该颜色的任何一个区域表现出来。这种选择与GIF系统的相对简便性相结合，使得在简单动画应用中GIF是一个合乎逻辑的选择，这种动画应用中的多重图像必须在计算机屏幕上移动。另一方面，它只能够对256种颜色编码，这就使得它不适合需要高精确度的应用，如摄影领域。

另外一种流行的图像压缩系统是JPEG（读作“JAY-peg”）。它是由ISO中的**联合图像专家组**（Joint Photographic Experts Group，标准因此得名）研制开发的标准。JPEG已经被证实是压缩彩色照片的一种有效标准，并被广泛用于摄影业。事实表明，大多数数码相机都将JPEG作为它们默认的压缩技术。

JPEG标准实际上包含多种图像压缩方法，每种都有它自己的目标。在需要绝对精确的情况下，JPEG可提供无损模式。不过，相对于JPEG的其他模式，JPEG的无损模式不能形成高级别的压缩，而且JPEG的其他选择模式已经很成功，这就意味着人们很少使用其无损模式。相反，称为JPEG基线标准的选择模式（也称为JPEG的有损顺序模式）已经成为许多应用的选择标准。

使用JPEG基线标准的图像压缩有几个步骤，其中有一些是利用人眼的局限性设计的。尤其是，相对于颜色的变化，人眼对亮度的变化更为敏感。因此，我们首先看一幅用光照和色度编码的图像。第一步，在一个2×2的像素方块中求色度的平均值，这样色度信息的大小减小为$\frac{1}{4}$，但保留了所有的原始亮度信息。结果在没有明显图像质量损失的情况下获得了很高的压缩率。

下一步是将图像拆分成8×8的像素块，然后将每一个块作为一个单元来压缩信息。这是通过一种称为离散余弦转换的数学技术实现的，我们现在不需要关心这个转换的细节。更重要的是，这种转换将原始的8×8块变成了另外一种块，其中的条目反映了原始块中像素之间如何相互联系，而并不是实际像素值。在这个块里，那些低于设定极限的数值将被0替代，反映的是这些数值所表示出的变化非常小，人眼无法觉察。例如，如果原始块中包含一个棋盘模式，那么新的块就可能表现为平均色。（典型的 8 × 8 像素块通常表示图像中一个非常小的方块，因此人眼根本不能够识别棋盘的外观。）

这时候，更传统的行程编码、相对编码以及变长编码技术被用于进一步压缩。总之，JPEG基线标准一般将彩色图像压缩至少10倍，有时甚至要30倍，而没有明显的质量损失。

另外一个图像数据压缩系统是TIFF（Tagged Image File Format的缩写）。不过，TIFF最普遍的应用不是数据压缩，而是存储照片的一个标准格式，同时要存储上相关的信息，如日期、时间以及相机设置。这时候，图像本身通常被存储为没有压缩的红、绿和蓝像素成分。

TIFF标准组合的确包含数据压缩技术，大多数是为在传真应用中压缩文本文档的图像设计的。它们使用行程编码的变体，目的是利用文本文件包含白色像素的长位串这一事实。TIFF标准中的彩色图像压缩选择以类似于GIF所使用的技术为基础，因此并没有被广泛应用于摄影业。

1.8.3 音频和视频压缩

音频及视频的编码和压缩最常用的标准是由ISO领导的MPEG（Motion Picture Experts Group，运动图像专家组）研制开发的，因此这些标准称为MPEG。

MPEG包含许多不同应用的许多标准。例如，高清晰电视（HDTV）广播的要求与视频会议的就不同，视频会议中播音信号必须经由可能容量有限的传输通道。另外，这两种应用又都不同于存储视频，它的有些部分可以被重放或略过。

MPEG使用的技术已经超出了本书的范围。但是一般说来，与存储动画到胶片上基本相同，视频压缩技术基于由一系列图片构建成的视频。为了压缩这些序列，只有一部分图片，称为I帧（I-frame），是被整个编码的。在I帧之间的图片采用相对编码技术。也就是说，并没有对整个图片编码，只是将与前一幅图不同的地方编码。I帧本身经常使用类似于JPEG的技术压缩。

压缩音频的最著名系统是MP3，它是在MPEG标准中开发出来的。事实上，MP3是MPEG layer 3的缩写。与其他压缩技术相比，MP3利用人耳的特性，删除了人耳觉察不到的细节。其中一个特性称为**暂时模糊**（temporal masking），指的是在一次巨大声响后，短时间内人耳觉察不到本可以听见的轻柔声音。另一个称为**频率模糊**（frequency masking），指的是某一频率的声音可能掩盖相近频率的轻柔声音。利用这些特性，MP3就可以获得视频的有效压缩，而且音质接近CD。

使用MPEG和MP3压缩技术，摄影机用128 MB的存储空间就可以录制长达1小时的视频，而且便携音乐播放器在1GB里就可以存储多达400首流行歌曲。但是，不同于其他压缩目的，音频和视频的压缩不一定需要节省存储空间。真正重要的是获得编码，使得信息能够通过今天的通信系统得到及时的传输。如果每一个视频帧需要1MB的存储空间，而且传播帧的通信路径每秒钟只能传输1 KB，那么根本无法实现成功的视频会议。因此，除了被认可的复制质量，音频和视频压缩系统的鉴定还有赖于实时数据传输的速率。这些速率通常用**bit/s**（bit per second，比特/秒）来度量。基本的单位包括**Kbit/s**（kilo-bps，等于1000 bit/s），**Mbit/s**（mega-bps，等于10^6bit/s），**Gbit/s**（giga-bps，等于10^9bit/s）。使用MPEG技术，视频展示可以通过40 Mbit/s的传输速率被成功传输。MP3录制需要的传输速率一般不超过64 Kbit/s。

问题与练习

1. 列出4种通用的压缩技术。
2. 使用LZW压缩，字典最初为x、y和一个空格（如文中所述），那么信息
   xyx yxxxy xyx yxxxy yxxxy
   如何编码？
3. 对彩色卡通编码时，为什么GIF比JPEG要好？

4. 假设你参与设计太空船，它要驶向其他星球并发回照片。那么，为了减少存储和传输图像的资源，使用GIF或JPEG的基准标准压缩照片是否是一个好主意？
5. JPEG的基准标准利用了人耳的什么特性？
6. MP3利用了人耳的什么特性？
7. 说出把数字信息、图像和声音编码为位模式时一种常见的麻烦现象。

*1.9 通信差错

当信息在计算机的各部分之间来回传输，或在月球和地球之间来回传输，又或者只是被保存在存储器中，最终检索到的位模式有可能和最原始的不一致。灰尘粒、磁盘表面的油脂或者一个出了故障的电路都可能使数据被错误地记录或读取。传输过程的静电干扰可能会损坏一部分数据。另外，在某些技术条件下普通的隐蔽放射线可以改变计算机主存储器中的存储模式。

为了解决这样的问题，人们开发了许多编码技术来检测甚至校正错误。今天，由于这些技术被大规模地内置于计算机系统的内部构件，计算机使用者并不了解它们。不过，它们是很重要的，为科学研究作出了很大的贡献。因此，我们有必要了解一些使计算机设备可靠的技术。

1.9.1 奇偶校验位

一种简单的错误检测方法基于下面的原则，即如果被操作的每个位模式都有奇数个1，但却找到了有偶数个1的模式，那么一定是出错了。使用这个原则，我们需要这样一个编码系统，其中每个模式有奇数个1。这是很容易做到的，首先在编码系统已经可用的模式（也许在高位端）上添加一位，称为**奇偶校验位**（parity bit）。在任何情况下，我们给这个新的位赋值1或0，这样整个模式就有奇数个1。一旦我们这样调整了编码系统，有偶数个1的模式就表示出现了错误，被操作的模式也是不正确的。

图1-28向我们展示了如何将奇偶校验位加到字母A和F的ASCII码上。注意，A的编码变成了101000001（奇偶校验位为1），F的ASCII码变成了001000110（奇偶校验位为0）。尽管A原始的8位模式有偶数个1，F原始的8位模式有奇数个1，但它们的9位模式都有奇数个1。如果将这种技术应用于所有的8位ASCII模式，我们就能够得到一个9位编码系统，其中任何一个9位模式有偶数个1就表明出错了。

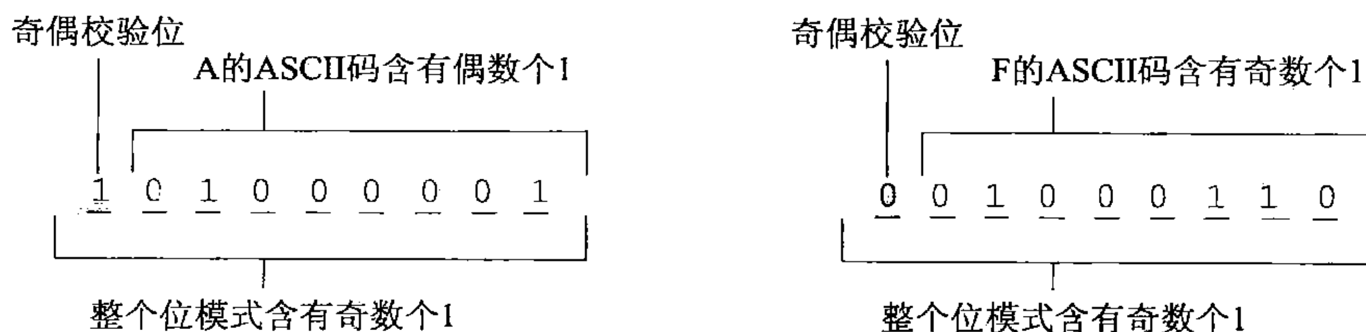


图1-28 适用奇校验的字母A和F的ASCII码

上面描述的奇偶校验系统称为**奇校验**，因为我们设计的系统使得每一个正确的模式都有奇数个1。另一种技术称为**偶校验**。在一个偶校验系统中，每个模式都被设计成包含偶数个1，因此如果出现了奇数个1，那么就表明有错误。

今天，在计算机主存储器中使用奇偶校验位已经不是一件稀奇的事了。尽管我们假设这些计算机存储单元是8位的，但事实上，它们可能是9位，其中一个位被用作奇偶校验位。每次传输一个8位模式给存储电路存储，电路都会给其加上一个校验位，存储结果为9位模式。在后来

检索模式时，电路检验这个9位模式的奇偶性。如果这样没有发现错误，存储器就移走校验位，然后自信地返回余下的8位模式。否则，存储器返回那8个数据位，并警告返回的模式可能与原本传给存储器的模式不同。

直接使用校验位很简单，但是有其局限性。如果一个模式最初有奇数个1，并出现了2次错误，那么它就仍然有奇数个1，这样校验系统就无法发现其错误。事实上，直接使用校验位不能发现模式中任何偶数个错误。

有时对长位模式应用一种方法来减少这类问题，例如磁盘扇区存储的位串。这种情况下，模式都有一组校验位构成的**校验字节**（checkbyte）。校验字节中的每一个位是一个校验位，与散布于整个模式中的一组特殊位相联系。例如，一个校验位可能与该模式中从第一个位起的每个第8位相关联，而另一个与该模式中从第二位起的每个第8位相关联。这样，集中在原模式某个区域中的一组差错就很可能被发现，因为它会在一些校验位的范围内。这个校验字节概念的演变引出了称为**校验和**（checksum）及CRC（Cyclic Redundancy Check，循环冗余校验）的差错检测方案。

1.9.2 纠错编码

虽然使用校验位可以发现差错，但是它不能提供纠正那个差错所需的信息。难怪设计出既能够发现差错又能纠正差错的**纠错码**（error-correcting code）会令很多人感到惊讶。毕竟直觉告诉我们，如果不知道信息的内容就无法纠正接收信息中的错误，但图1-29给我们展示了一个具有这样纠错特性的编码。

为了明白这个编码是如何运作的，我们先定义**汉明距离**（Hamming distance，根据R. W. Hamming的姓氏命名，由于在20世纪40年代继电器可靠性方面备受挫折，他开始开创性地研究起纠错码来）。两个位模式之间的汉明距离指的是这两个模式中不相同位的个数。例如，图1-29编码中表示A和B模式的汉明距离是4，而B和C间的汉明模式是3。图1-29编码中最重要的特征是，任何两个模式之间的汉明距离至少是3。

如果用图1-29的模式修改单个位，就会发现错误，因为它的结果不会是一个合法的模式。（我们至少要将每个模式改变3个位，这样它们才会像另外一个合法模式。）而且，我们能够指出原始模式是什么。毕竟，修改过的模式和其原始形式的汉明距离是1，而和其他任何合法模式的汉明距离至少是2。

因此，对最初用图1-29编码的信息解码，我们只需要对比接收模式和用此编码表示的模式，直到我们找到一个和接收模式之间的汉明距离是1的模式为止。我们将其视为正确的符号进行解码。例如，我们接收到位模式010100，然后将其与用编码表示的模式相比，我们就会获得图1-30中所示的表格。因此，我们得出结论，传输的字符一定是D，因为这是最接近的匹配。

符号	代码
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

图1-29　一个纠错码

字符	编码	接收到的模式	接收到的模式与编码之间的距离
A	0 0 0 0 0 0	0 1 0 1 0 0	2
B	0 0 1 1 1 1	0 1 0 1 0 0	4
C	0 1 0 0 1 1	0 1 0 1 0 0	3
D	0 1 1 1 0 0	0 1 0 1 0 0	1 ——— 最小距离
E	1 0 0 1 1 0	0 1 0 1 0 0	3
F	1 0 1 0 0 1	0 1 0 1 0 0	5
G	1 1 0 1 0 1	0 1 0 1 0 0	2
H	1 1 1 0 1 0	0 1 0 1 0 0	4

图1-30　用图1-29的编码对模式010100解码

可以发现，使用图1-29的编码技术，每个模式我们可最多检测出2个错误并改正1个。如果我们能设计出这样一种编码，每个模式和其他任何模式之间的汉明距离都至少是5，每个模式我们就将最多发现4个错误，并最多改正2个。当然，设计一种具有长汉明距离的编码不是一件简单的事。事实上，它是称为代数编码理论的数学分支的一部分，这个理论是线性代数和矩阵理论的子领域。

纠错技术被广泛用于增加计算设备的可靠性。例如，它们经常被用于高容量磁盘驱动器，以减少因磁盘表面瑕疵而损坏数据的可能性。此外，最初用于音频的CD格式与后来用于计算机数据存储的格式之间的主要差别是纠错的程度。CD-DA格式包括的纠错特点使得错误率减少到2张CD只有1个错误。这对视频录制足够了，但是对于用CD向客户交付软件的公司，若50%的磁盘有瑕疵是令人无法忍受的。因此，人们将附加的纠错特征用在CD中来存储数据，使产生错误的可能性减少到20 000个磁盘1个错误。

问题与练习

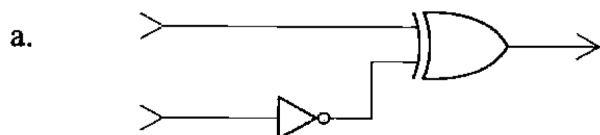
1. 下面的字节最初是用奇校验编码的。找出出错的一个。
   a. `100101101` b. `100000001` c. `000000000` d. `111000000` e. `011111111`
2. 问题1中没发现错误的字节还会有错误吗？解释原因。
3. 如果将奇校验换成偶校验，那么问题1和问题2的答案又将如何？
4. 用带奇校验的ASCII码对这些语句编码，在每一个字符编码的高位端加一个校验位。
   a. “Stop!” Cheryl shouted.
   b. Does 2+3=5?
5. 用图1-29的纠错码对下面的信息解码。
   a. `001111 100100 001100`
   b. `010001 000000 001011`
   c. `011010 110110 100000 011100`
6. 用长度为5的位模式，为字符A、B、C和D建造编码，使得任何两个模式之间的汉明距离至少是3。

复习题

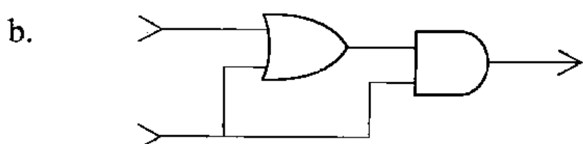
（带*的题目涉及选读小节的内容。）

1. 假设上面输入为1下面输入为0，请确定下面每一个电路的输出值。如果上面输入为0，下面输入为1呢？

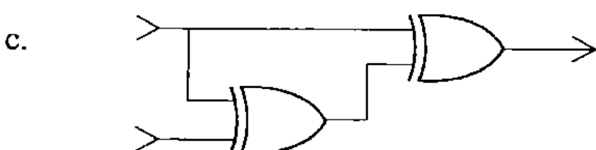
a.



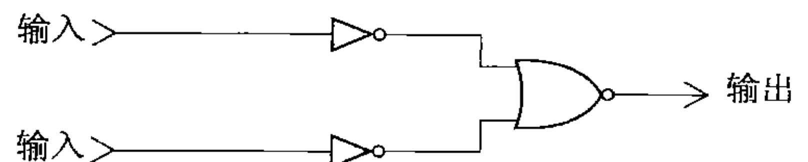
b.



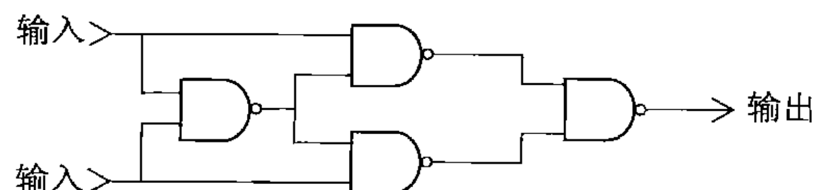
c.



2. a. 下面这个电路用来完成什么布尔运算？

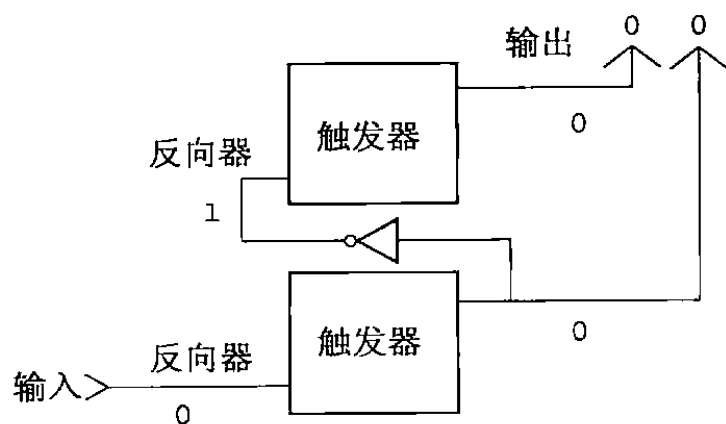


b. 下面这个电路用来完成什么布尔运算？

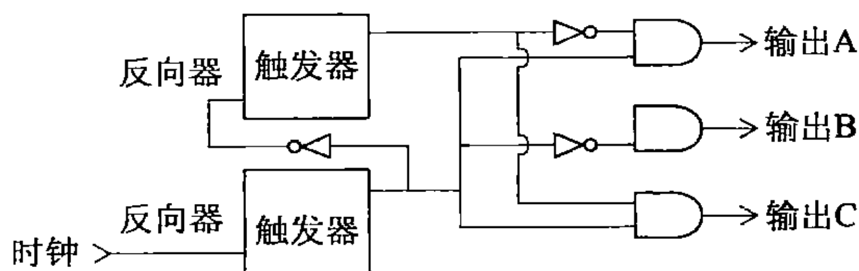


*3. a. 如果要从某个电子元件商店购买触发器电路，我们会发现它有一个额外的输入端，称为反向器（flip）。当反向器的输入从0变成1时，输出将翻转状态（如果原来是0，现在就是1，反之亦然）。但是，当翻转输入从1变为0时，什么都不会发生。虽然我们可能不知道电路完成这一行为所需的细节，但仍然能够在其他电路中将这一设备

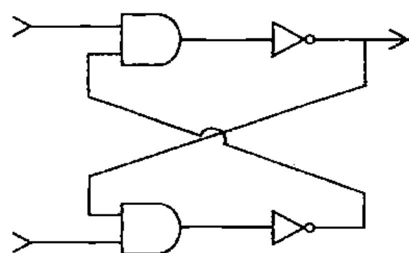
作为一个抽象工具来使用。请考虑使用下面两个触发器的电路。如果在电路的输入端发送一个脉冲，下面的那个触发器将变换状态。但是，另一个触发器不会改变，因为其输入（这一输入是从非门的输出接收到的）从1变成了0。因此，这一电路现在的输出为0和1。在电路输入端发送第二个脉冲将翻转两个触发器的状态，并产生输出1和0。如果在电路输入端发送第三个脉冲，将会产生什么输出呢？发送第四个脉冲呢？



b. 通常有必要协调一台计算机中不同组件的活动。只需将脉冲信号（称为时钟）连接到类似a中的电路即可。额外的门（如图所示）将以协同的方式发送信号到其他连接的电路。研究这一电路时，你将能够确认这样一个事实，即在第一次、第五次、第九次（依次类推）时钟脉冲下，输出端A将发送1。哪些时钟脉冲将使输出端B发送1？哪些时钟脉冲将使输出端C发送1？第四次时钟脉冲时，哪个输出端为1？



4. 假设下面电路的两个输入都是1。请描述如果上面输出暂时变为0，会发生什么？如果下面输入暂时变为0，又会发生什么？用与非门重新绘制这个电路。



5. 下面表格表示的是计算机主存储器某些单元的地址和内容（采用十六进制记数法）。首先按照这个存储安排，遵循下列指令，记录下这些存储单元的最后内容。

地　址	内　容
00	AB
01	53
02	D6
03	02

步骤1：将地址为03的单元的内容移动到地址为00的单元中。

步骤2：将数值01送到地址为02的单元。

步骤3：将存储在地址01中的数值移动到地址为03的单元。

6. 如果每个单元地址用2个十六进制数字表示，那么一台计算机的主存储器中可以有多少个单元？如果用4个十六进制数字呢？

7. 什么位模式可以用下面的十六进制记数法表示？

a. CD　b. 67　c. 9A　d. FF　e. 10

8. 下面十六进制记数法表示的位模式中，最高有效位的数值是什么？

a. 8F　b. FF　c. 6F　d. 1F

9. 用十六进制记数法表示下面的位模式。

a. 101000001010　b. 110001111011

c. 000010111110

10. 假设一个数码相机的存储容量是256 MB。如果每个像素需要3个字节的存储空间，而且一张照片包括每行1024像素及每列1024像素，那么这台数码相机可以存多少张照片？

11. 假设一张图片以1024列及768行像素的矩形形式显示在计算机屏幕上。如果对于每个像素需要8位来对颜色编码，并用另外8位对亮度编码，那么整幅图片需要多少字节的存储单元？

12. a. 指出主存储器优于磁盘存储的两个优点。

b. 指出磁盘存储优于主存储器的两个优点。

13. 假设个人计算机上120GB的硬盘只剩下50 GB是空闲的，那么用CD备份硬盘上的资料是否合理?用DVD呢?

14. 如果磁盘的每一个扇区包含1024个字节，那么如果每个字符用Unicode表示，存储一页文本（如50行，每行100个字符）需要多少扇区?

15. 用ASCII码，每页3500个字符，存储一本400页的小说需要多少字节的存储空间？如果用Unicode又需要多少字节?

16. 一个硬盘驱动器每秒钟转360转，那么它的等待时间是多少?

17. 如果一个硬盘驱动器每秒转360转，寻道时间是10 ms，那么它的平均存取时间是多少?
18. 如果一个打字员每天24小时打字，每分钟打60个字，那么这个打字员要多久能填满容量是640 MB的CD？假定一个单词是5个字符，每个字符需要1个字节的存储空间。
19. 下面是用ASCII编码的信息。内容是什么?

    01010111 01101000 01100001 01110100 00100000 01100100
    01101111 01100101 01110011 00100000 01101001 01110100
    00100000 01110011 01100001 01111001 00111111
20. 下面信息使用ASCII编码，每个字符一个字节，并用十六进制记数法表示出来。内容是什么？

    68657861646563696D616C
21. 用ASCII对下面的句子编码，每个字符一字节。
    a. Does 100/5=20?
    b. The total cost is $7.25.
22. 将前面问题的答案用十六进制记数法表示出来。
23. 列出整数8～18的二进制表示。
24. a. 分别用ASCII码表示2和3，写出数字23。
    b. 用二进制表示写出数字23。
25. 什么值的二进制表示只有一个位为1？列出具有这个特性的最小的6个值的二进制表示。

*26. 将下面的每个二进制表示转换成相应的十进制表示。

    a. 1111　b. 0001　c. 10101
    d. 1000　e. 10011　f. 000000
    g. 1001　h. 10001　i. 100001
    j. 11001　k. 11010　l. 11011

*27. 将下面每个十进制表示转换成相应的二进制表示。

    a. 7　b. 11　c. 16　d. 17　e. 31

*28. 将下面的每一个余16表示转换成相应的十进制表示。

    a. 10001　b. 10101　c. 01101
    d. 01111　e. 11111

*29. 将下面的每一个十进制表示转换成相应的余4表示。

    a. 0　b. 3　c. −2　d. −1　e. 2

*30. 将下面的每个二进制补码表示转换成相应的十进制表示。

    a. 01111　b. 10100　c. 01100
    d. 10000　e. 10110

*31. 将下面的每个十进制表示转换成相应的二进制补码表示，其中每个值用7位表示。

    a. 13　b. −13　c. −1　d. 0　e. 16

*32. 假定下面这些位串都是用二进制补码记数法表示的值，执行下面这些加法运算。辨别哪一个由于溢出而答案不正确。

    a. 00101 +01000　b. 11111 +00001　c. 01111 +00001
    d. 10111 +11010　e. 11111 +11111　f. 00111 +01100

*33. 解答下面的每个问题：将这些值翻译成二进制补码记数法（用5位模式），转换任何一个减法运算为相应的加法运算并执行。将所得答案转换成十进制记数法进行验证。（观察溢出现象。）

    a. 5 +1　b. 5 −1　c. 12 −5
    d. 8 −7　e. 12 +5　f. 5 −11

*34. 将下面的每个二进制表示转换成相应的十进制表示。

    a. 11.11　b. 100.0101　c. 0.1101
    d. 1.0　e. 10.01

*35. 用二进制记数法表示下面每个值。

    a. $5\frac{3}{4}$　b. $15\frac{15}{16}$　c. $5\frac{3}{8}$
    d. $1\frac{1}{4}$　e. $6\frac{5}{8}$

*36. 用图1-26所示的浮点格式对下面的位模式解码。

    a. 01011001　b. 11001000　c. 10101100
    d. 00111001

*37. 用图1-26所示的8位浮点格式对下面的值编码。指出出现截断误差的每个情形。

    a. $-7\frac{1}{2}$　b. $\frac{1}{2}$　c. $-3\frac{3}{4}$
    d. $\frac{7}{32}$　e. $\frac{31}{32}$

*38. 假定你不受使用标准化格式的限制，用图1-26所示的浮点格式列出所有可以表示数值 $\frac{3}{8}$ 的位模式。

*39. 用图1-26所示的8位浮点格式，求可以表示的最近似于2的平方根的值。如果计算机利用这

一浮点格式对这个数做平方，实际得到的值是什么？

*40. 用图1-26所示的8位浮点格式可以表示的最近似于$\frac{1}{10}$的数值。

*41. 当米制系统的度量用浮点记数法记录时，解释为什么会产生误差？例如，若110cm用米制单元记录情况会怎样？

*42. 位模式01011和11011表示同一个值，一个用余16记数法存储，另外一个用二进制补码记数法存储。

a. 共同表示的这个值是什么？

b. 同一个值分别用二进制补码记数法和余码记数法存储，并且这两个系统采用相同的位模式长度，那么表示此值的这两种模式是一种什么关系？

*43. 3个位模式10000010、01101000和00000010表示同一个值，分别是采用二进制补码记数法，余码记数法和图1-26所示的8位浮点格式，但并不是必须按照上面顺序一一对应。那么它们共同表示的值是什么？哪个模式对应哪个记数法？

*44. 在下面的值中，哪一个不能用图1-26所示的浮点格式精确地表示出来？

a. $6\frac{1}{2}$ b. $\frac{13}{16}$ c. 9 d. $\frac{17}{32}$ e. $\frac{15}{16}$

*45. 用二进制表示整数的位串长度由4变成6，那么能够表示的最大整数值会发生什么变化？用二进制补码记数法又将如何呢？

*46. 一个存储器容量是4 MB，每个单元可以存储1字节，那么其最大地址的十六制表示是什么？

*47. 使用LZW压缩，并且最初的字典是x、y和一个空格（如1.8节所述），那么下面的信息如何编码？

xxy yyx xxy xxy yyx

*48. 下面信息使用LZW压缩，其字典的第1、2和3个条目分别是x、y和空格。解压缩这条信息：

22123113431213536

*49. 如果信息

xxy yyx xxy xxyy

用LZW压缩，最初字典的第1、2和3个条目分别是x、y和空格。那么最后的字典条目是什么？

*50. 我们将在下一章学到，通过传统电话系统传输位的一种方法是，首先将位模式转换成声音，通过电话线传输声音，接着再将声音转换成位模式。这种技术的传输速率最多可达57.6 Kbit/s。那么如果视频采用MPEG压缩，这是否能满足远程会议的需要？

*51. 使用ASCII码对下面的句子编码，并使用偶校验。在每个字符编码的高位端添加一个校验位。

a. Does 100/5=20?

b. The total cost is $7.25.

*52. 下面信息的每一个短位串，最初都是用奇校验传输的。那么哪一个位串绝对出现了差错？

11001 11011 10110 00000 11111 10001 10101 00100 01110

*53. 假如一个24位的编码是这样产生的：复制3个符号连续的ASCII编码，并用它们表示每个符号（例如，符号A用位串010000010100000101000001表示）。这个新编码有哪些纠错特性？

*54. 用图1-30的纠错码对下面的词语解码。

a. 111010 110110

b. 101000 100110 001100

c. 011101 000110 000000 010100

d. 010010 001000 001110 101111 000000 110111 100110

e. 010011 000000 101001 100110

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答出这些问题还不够，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 某个截断错误出现在一个关键时刻，引起了巨大的损失和人身伤亡。如果有人需要对此负责，那么是谁？是硬件设计者，软件设计者，编写那段程序的程序员，还是决定在那个特定应用中使用这个软件的人？如果最初设计这个软件的公司已经修正过这个软件，但是用户还没有购买这个升级版就用于这个关键应用中，又将如何？如果这个软件是盗版的，又将如何？

2. 一个人开发的应用忽略了截断错误的可能性及它们的后果，这是可以接受的吗?
3. 如果在20世纪70年代只用2个数字开发表示年的软件（例如用76表示1976年），而忽视了这个软件在即将到来的世纪之交会失效，这符合伦理道德吗？若今天只用3个数字表示年（例如用982表示1982，用015表示2015）又是否符合伦理呢？如果只用4个数字呢?
4. 许多人认为，对信息进行编码经常会削弱或歪曲该信息，因为这实质上迫使信息必须被量化。他们认为，若一份调查问卷每题给出了5个等级，并要求回答者按照此标准表达意见，那这份问卷本身就是无效的。信息可以量化到什么程度？垃圾处理场选址的利弊可以量化吗？关于核能源及核废料的辩论是可量化的吗？将结论建立于平均值和其他统计分析上的做法危险吗？如果新闻通讯社在报导调查结果时不使用问题的准确措词，这合乎道德吗？能够量化一个人的生命值吗？假设一个公司要停止对一个产品改进的投资，尽管知道附加投资可以减少产品使用的危险性，这合理吗？
5. 在收集和散发数据的权利上，是否根据数据的形式有所差别？也就是说，收集和散发照片、音频或者视频的权利是否与收集和散发文本一样？
6. 无论有意还是无意，记者的报道通常反映了自己的倾向。通常只要改几个词语，一篇报道就可能被赋予正面或负面的含义。(比较:“大多数被调查的人都反对公民投票权”,“被调查人中有相当一部分支持公民投票权”。) 修改一篇报道（回避某些观点或者仔细选词）和修改一张照片有区别吗？
7. 假设用数据压缩系统后导致一些微小但很重要的信息丢失了。这会产生什么样的责任问题？怎么解决？

课外阅读

Drew, M. and Z. Li. *Fundamentals of Multimedia*. Upper Saddle River, NJ: Prentice-Hall, 2004.

Halsall, F. *Multimedia Communications*. Boston, MA: Addison-Wesley, 2001.

Hamacher, V. C., Z. G. Vranesic, and S. G. Zaky. *Computer Organization*, 5th ed. New York: McGraw-Hill, 2002.

Knuth, D. E. *The Art of Computer Programming*, vol. 2, 3rd ed. Boston, MA: Addison-Wesley, 1998.

Long, B. *Complete Digital Photography*, 3rd ed. Hingham, MA: Charles River Media, 2005.

Miano, J. *Compressed Image File Formats*. New York: ACM Press,1999.

Petzold, C. *CODE: The Hidden Language of Computer Hardware and Software*.

Redman, WA: Microsoft Press, 2000.

Salomon, D. *Data Compression: The Complete Refernce*, 4th ed. New York: Springer, 2007.

Sayood, K. *Introduction to Data Compression*, 3rd ed. San Francisco: Morgan Kaufmann, 2005.

第2章 数据操控

本章学习计算机如何操控数据以及如何与外围设备（如打印机和键盘）通信。为此，我们将研究计算机体系结构的基础知识，学习计算机是如何利用称为机器语言指令的编码指令来进行编程工作的。

本章内容

在第1章中，我们学习了有关计算机数据存储的主题，本章将介绍计算机如何操控这些数据，其内容包括数据在不同位置间的移动，以及诸如算术计算、文本编辑和图像处理等的操作。首先我们要了解除数据存储系统之外的计算机体系结构。

2.1 计算机体系结构

计算机中控制数据操控的电路称为CPU（Central Processing Unit，中央处理器，通常简称为处理器）。在20世纪中期，CPU属于大部件，由若干机架中的电子线路组成，这也反映了该部件的重要性。不过，科技进步已经极大地缩小了这些部件。今天，PC机和笔记本电脑中的CPU都是很小的正方形薄片（大约都只有2×2英寸），它们的引脚插在计算机主电路板［称为**主板**（motherboard)］的插座上。在智能手机、小型笔记本电脑和其他MID（Mobile Internet Device，移动因特网设备）上，CPU大约是邮票的一半大小。由于它们的尺寸比较小，这些处理器被称作**微处理器**（microprocessor）。

2.1.1 CPU基础知识

CPU由3部分构成（图2-1）：**算术/逻辑单元**（arithmetic/logic unit），它包含在数据上执行运算（如加法和减法）的电路；**控制单元**（control unit），它包含协调机器活动的电路；**寄存器单元**（register unit），它包含称为**寄存器**（register）的数据存储单元（与主存单元相似），用作CPU内部的信息临时存储。

寄存器单元中的一些寄存器被看成是**通用寄存器**（general-purpose register），而其他一些则被看成是**专用寄存器**（special-purpose register），我们将在2.3节讨论一些专用寄存器，现在我们只关注通用寄存器。

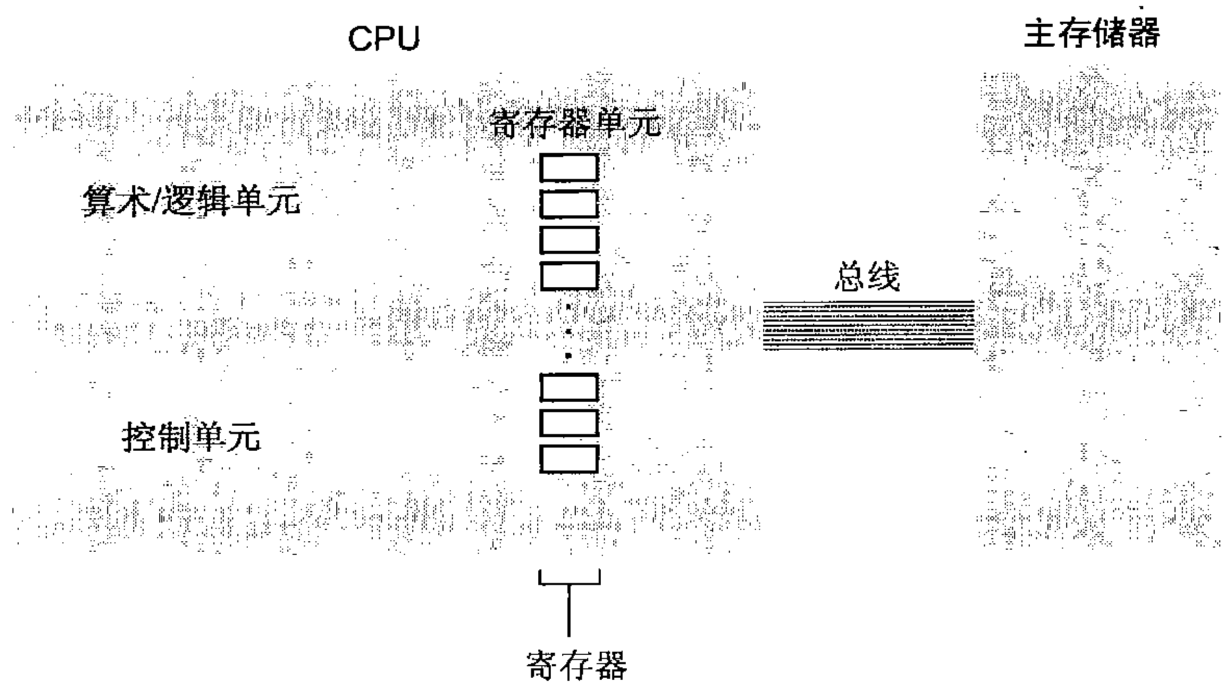


图2-1 通过总线连接的CPU和主存储器

通用寄存器用于临时存储CPU正在操控的数据。这些寄存器存储算术/逻辑单元电路的输入值以及该部件所产生的结果。为了操作存储在主存储器中的数据，控制单元要把存储器里的数据传送到通用寄存器，通知算术/逻辑单元由哪些寄存器保存了这一数据，激活算术/逻辑单元中的有关电路，并告知算术/逻辑单元哪个寄存器将接收结果。

为了传输位模式，计算机CPU和主存储器通过一组称为**总线**（bus，图2-1）的线路进行连接。利用总线，CPU给出相关存储单元的地址以及相应的电信号（告知存储器电路，将在指定单元中获取数据），从主存储器中取（读）出数据，同理，CPU可以向主存储器中放入（写入）数据，方法是提供目的单元地址和一起写入的数据以及适当的电信号（告知主存储器，将要储存发送给它的数据）。

基于此设计，将存储在主存储器中的两个值相加的任务不仅仅涉及执行加法运算。数据必须先从主存储器传输到CPU的寄存器中，值相加后，把结果放置在寄存器中，然后再把结果存储到主存单元中。整个过程被总结成如图2-2所示的5个步骤。

步骤 1 从存储器中取出一个要加的值放入一个寄存器中。
步骤 2 从存储器中取出另一个要加的值放入另一个寄存器中。
步骤 3 激活加法电路，以步骤1和2所用的寄存器作为输入，用另一个寄存器存放相加的结果。
步骤 4 将结果存入存储器。
步骤 5 停止。

图2-2 主存储器中的值相加

2.1.2 存储程序概念

早期计算机不是很灵活，每个设备所执行的步骤都被内置于控制单元中，作为计算机的一部分。为了增加其灵活性，早期电子计算机的设计使得CPU可以方便地重新布线。其灵活性通过插拔装置体现，类似于老式的电话交换台上把跳线的端子插到接线孔中。

认识到一个程序可以像数据一样进行编码并存储在主存储器中，这是一个突破性进展（但是将其归功于约翰·冯·诺依曼显然是不正确的）。如果控制单元可以从存储器中读取程序、将指令解码并执行它们，那么机器要遵照执行的程序就可以被修改，这只需要改变计算机存储器中的内容而不必对CPU进行重新布线。

高速缓冲存储器

将计算机存储设备与其对应功能进行比较是很有启发性的。寄存器用于存储可立即进行运算的数据，主存储器用于存储即将使用的数据，海量存储器用于存储最近也许不会使用的数据。许多计算机设计增加了一个附加的存储器层次，称为高速缓冲存储器。**高速缓冲存储器**（cache memory）是位于CPU内部的高速存储器的一部分（也许有几百KB）。在这个特殊的存储区域中，计算机试图保存主存储器中当前最重要的那部分内容的一个副本。这样，通常要在寄存器与主存储器之间进行的数据传输将变成寄存器与高速缓冲存储器之间的数据传输。因此，高速缓冲存储器中的任何改变都会在恰当时间被一起传输给主存储器。于是，CPU可以较快地执行它的机器周期，因为它不会被与主存储器的通信所延迟。

谁的发明

将某项发明的荣誉授予个人总是备受争议。人们将白炽灯的发明归功于托马斯·爱迪生（Thomas Edison），但是其他研究员也曾研制了类似的灯泡，从某种意义上说，爱迪生只是比较幸运地获得了专利。人们认为是莱特（Wright）兄弟发明的飞机，但他们曾与其他人竞争并受益于其他人的研究，在某种程度上，他们又被达芬奇抢先了，他早在15世纪就有了玩玩飞行机器的想法。甚至达芬奇的设计看起来也是假借前人的思想。当然，对于这些发明，被认定的发明人还是有权拥有被授予的荣誉的。但对于其他一些情况，历史上的有些荣誉授予似乎是不恰当的，例如存储程序的概念。毫无疑问，约翰·冯·诺依曼（John von Neumann）是一位卓越的科学家，理应为自己的许多贡献获得荣誉。但是，历史选择授予他荣誉的贡献是存储程序概念，但这一思想很显然是由宾夕法尼亚大学莫尔电子工程学院以埃克特（J. P. Eckert）为首的研究人员提出的。约翰·冯·诺依曼不过是第一个在著作中转述这一思想的人，因此计算机界选择他作为发明人。

将计算机程序存储在主存储器中的思想称为**存储程序概念**（stored-program concept），它已经成为今天所使用的标准方法。最初的困难源于人们将程序和数据视为不同的实体：数据存储在存储器中，而程序为CPU的一部分。于是就成了“只见树木，不见森林”的一个最好实例。人们很容易被老一套所束缚，如果今天仍不了解这一点，那么计算机科学的发展也许还会裹足不前。的确，科学中令人兴奋的部分是，新的思想不断为新的理论和新的应用开启大门。

问题与练习

1. 将计算机中一个存储单元的内容移到另一个存储单元，你认为需要哪些事件序列？
2. 如果将一个值写入存储单元中，那么CPU需要提供给主存储器电路什么信息？
3. 海量存储器、主存储器以及通用寄存器都是存储系统。它们在用法上有什么不同？

2.2 机器语言

为了应用存储程序概念，CPU被设计成可以识别二进制模式编码的指令。这组指令以及编码系统统称为**机器语言**（machine language）。使用此语言表达的指令称为机器级指令，更多人称其为**机器指令**（machine instruction）。

2.2.1 指令系统

一个典型CPU必须能够解码及执行的机器指令列表非常短。实际上，一旦计算机能够实现

某些基本但适当的任务，那么添加再多的特性也不会增加该计算机理论上的能力。换句话说，超过某个特定点之后，附加的特性也许能够增加诸如便利性等能力，但是不能增加该计算机的基本能力。

在设计计算机时，该事实将被利用到何种程度导致了两种CPU体系结构的出现。一方面是CPU只需要执行最小的机器指令集，因此产生了**RISC**（Reduced Instruction Set Computer，精简指令集计算机）。RISC的支持者认为，这样设计的计算机效率高、速度快，而且制造起来较便宜。另一方面，另外一些人则认为CPU应能够执行大量复杂的指令，尽管许多在技术上是多余的，因此产生了**CISC**（Complex Instruction Set Computer，复杂指令集计算机）。CISC的支持者认为，CPU越复杂越容易应对今天日益增加的软件复杂性。通过CISC，程序可以利用一组功能强大的丰富的指令集，其中很多指令所能实现的任务需要RISC设计中的一个多指令序列才能实现。

20世纪90年代至21世纪，市场上有售的CISC和RISC处理器在积极争抢桌面计算领域。用于个人计算机中的英特尔处理器是CISC体系结构的代表，而PowerPC处理器（由苹果、IBM和摩托罗拉公司联合开发）是RISC体系结构的代表，被用于苹果Macintosh中。随着时间的推移，CISC的制造成本大大降低了，因此英特尔的处理器（或AMD公司的处理器）现已几乎遍及台式机和笔记本电脑，甚至苹果公司生产的计算机也开始使用英特尔公司的产品。

虽然CISC在台式计算机领域站稳了脚跟，但其电耗非常大。与此相反，ARM（Advanced RISC Machine）公司专门针对低电耗设计了一种RISC体系节构。（ARM的前身是Acorn计算机公司，现在是ARM Holdings。）因此，ARM处理器（由多个供应商制造，包括高通和德州仪器）已出现在游戏控制器、数字电视、导航系统、汽车部件、移动电话、智能手机和其他消费性电子产品中。

不管选择RISC还是CISC，机器指令可以分为3类：（1）数据传输类；（2）算术／逻辑类；（3）控制类。

1．数据传输类

数据传输类指令包含请求在各个位置之间传输数据的指令，图2-2中的步骤1、2和4都属于这一类。需要注意的是，使用传输（transfer）或移动（move）来标识这组指令实际上是用词不恰当的。因为传输的数据很少被从原始位置擦除。执行传输指令的过程更像是复制数据而不是移动数据，因此，复制（copy）或克隆（clone）能更好地描述这组指令的活动。

关于术语，我们应注意到，提到CPU与主存储器之间数据的传输时，有专门的术语。用存储单元的内容填充通用寄存器的请求通常称为LOAD（加载）指令；相反，将寄存器中内容传输给存储单元的请求称为STORE（存储）指令。在图2-2中，步骤1和2是LOAD指令，步骤4是STORE指令。

在数据传输类中有这样一组重要的指令，即与CPU-主存储器环境之外的设备（打印机、键盘、显示器以及磁盘驱动器等）通信的指令。这些指令处理该机器的输入/输出活动（I/O），因此被称为**I/O指令**，且有时因为其特别而被单独归为一类。另一方面，2.5节将介绍这些I/O活动如何利用与请求在CPU及主存储器之间传输数据同样的指令来完成操作。因此，我们将I/O指令归入数据传输类指令。

2．算术/逻辑类

算术/逻辑类指令告诉控制单元请求在算术/逻辑单元内实现一个活动。图2-2中的步骤3属于这一类。正如其名称所示，算术/逻辑单元还能够执行基本算术运算之外的运算。在这些附加的运算中，就有第1章中介绍的布尔运算与、或和异或，本章后面将进一步讲解。

大多数算术/逻辑单元中可以用另外一组运算进行寄存器中内容的左右移动。这些运算称为移位（SHIFT）运算或循环移位（ROTATE）运算，前者丢弃一端“移出的位”，而后者将它们放到另一端留出的空位上。

3. 控制类

控制类指令包含指导程序执行而非数据操作的指令。图2-2中的步骤5属于此类，但它是一个很初级的例子。这一类包括计算机指令系统中许多比较有趣的指令，例如JUMP（转移）或BRANCH（分支）系列指令用于指示CPU执行非列表中下一条指令的指令。转移指令有两种，即**无条件转移**（unconditional jump）和**条件转移**（conditional jump）。前者的例子如“跳转到步骤5”，后者的例子如“如果所得数值为0，跳转到步骤5”。两者的区别是，只有满足某个条件时，条件转移才会引起“地点改变”。举例说明，图2-3中的指令序列是表示两个数值相除的算法，其中步骤3是条件转移，用以防止除数为0。

步骤 1　把存储器中的一个值加载到一个寄存器中。
步骤 2　把存储器中的另一个值加载到另一个寄存器。
步骤 3　如果第二个值为0，那么转移到步骤6。
步骤 4　第一个寄存器中的值除以第二个寄存器的值，结果留在第三个寄存器中。
步骤 5　把第三个寄存器的值存储到存储器。
步骤 6　停止。

图2-3　存储器中数值的除法运算

2.2.2 一种演示用的机器语言

现在让我们来分析一台典型的计算机的指令是如何编码的。我们用于讨论的计算机在附录C中描述，其体系结构见图2-4。它有16个通用寄存器，256个主存储器单元，每个存储单元容量为8位。为了便于参考，我们将寄存器标号为数值0~15，把存储单元的地址编为数值0~255。为方便起见，我们将这些标号及地址看做以二进制表示的数值，并使用十六进制记数法压缩它们的位模式。于是，寄存器标号为0~F，存储单元的地址为00~FF。

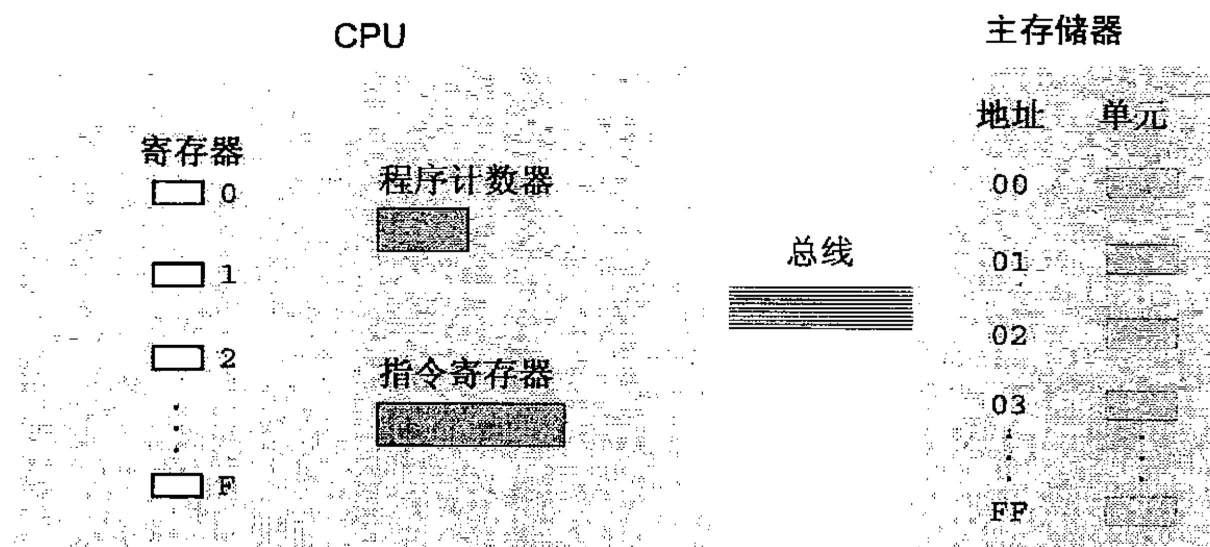


图2-4　附录C描述的计算机的体系结构

机器指令编码形式包括两部分，即**操作码**（operation code，缩写为op-code）字段和**操作数**（operand）字段。操作码字段中的位模式指明该指令要求的是什么基本运算，如STORE、SHIFT、XOR和JUMP等。操作数字段中的位模式提供操作码指定运算的更详细信息。以STORE操作为例，其操作数字段中的信息指示哪个寄存器包含将被存储的数据，哪个存储单元用于接收该数据。

我们演示用的计算机（见附录C）的整个机器语言只包含12条基本指令。每条指令都用16位编码，由4个十六进制数字表示（见图2-5）。每条指令的操作码由前4位组成，等价于第一个十六进制数字。注意（见附录C），这些操作码用十六进制数字1~C表示。特别是，附录C中的表说明以十六进制数字3起始的指令表示STORE指令，以十六进制A起始的指令表示ROTATE指令。

变长指令

为了简化文中的解释，本章示例所用的机器语言（在附录C中有描述）对于所有指令都使用了固定的大小（2个字节）。因此，为取得一个指令，CPU总是需要检索2个连续存储单元的内容，然后给其程序计数器加2。这种连贯性简化了取指令的任务，是RISC机器的特性。然而，对于CISC机器的机器语言，指令长度可变。例如，今天的英特尔处理器指令长度有大有小，小的只有1字节，长的要多个字节，这取决于该指令的确切应用。使用这样机器语言的CPU根据指令操作码来确定所引入指令的长度。也就是说，CPU首先取指令的操作码，然后基于收到的位模式得知，要得到余下的指令还需要从存储器中取多少字节。

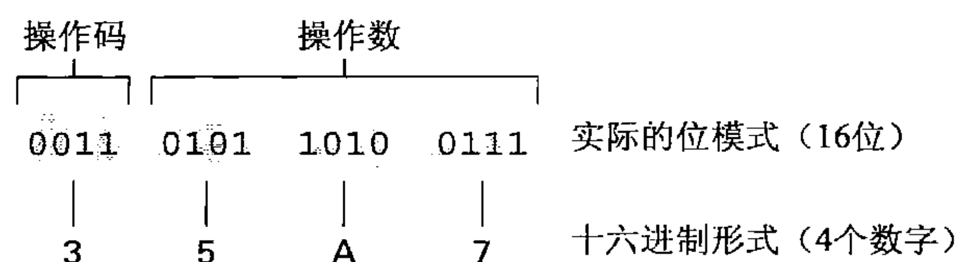


图2-5 一条指令的组成（附录C描述的计算机）

在我们演示用的计算机中，每条指令的操作数字段由3个十六进制数字（12位）组成，在每种情况下对操作码给定的通用指令做了进一步澄清（HALT指令除外，因为它不需要进一步的规定）。例如（见图2-6），如果一条指令的第一个十六进制数字为3（存储寄存器中内容的操作码），那么该指令的下一个十六进制数字则指出哪个寄存器中的内容需要存储，而最后的两个十六进制数字则指出由哪个存储单元接收该数据。因此，指令35A7（十六进制数）是指“将第5寄存器中的位模式存储（STORE）到地址为A7的存储单元”。（注意使用十六进制记数法是如何简化我们的讨论的。事实上，指令35A7是指位模式0011010110100111。）

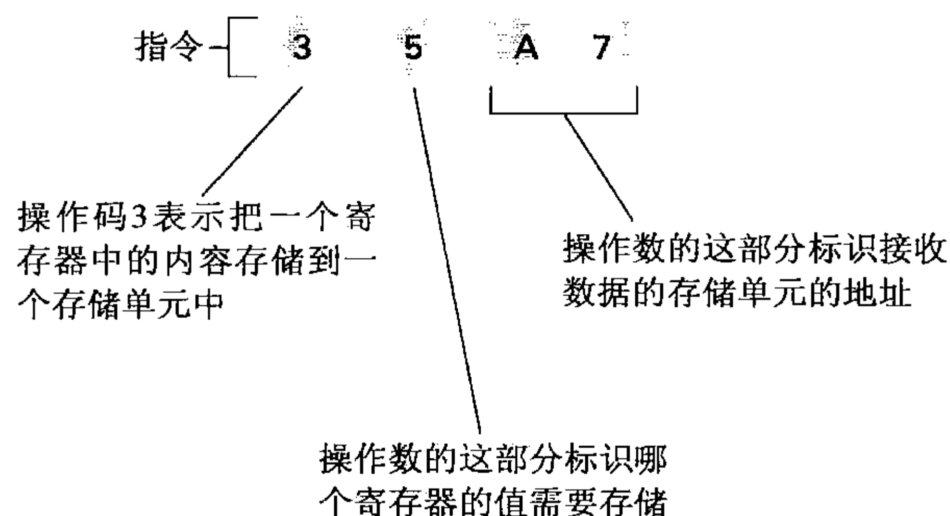


图2-6 指令35A7的译码

（对于主存储器容量为什么以2的幂为度量单位，指令35A7同样提供了一个明晰的例子。因为该指令保留8位用于指定该指令所用的存储单元，所以能够准确地引用2^8个不同的存储单元。因此我们需要用这么多存储单元构建1个主存储器——地址从0~255。如果主存储器有更多的存储单元，我们就不能够写出指令以区别它们；如果主存储器的存储单元比较少，我们写出的指令就可能引用不存在的存储单元。）

再举一个例子说明操作数字段如何阐明操作码给定的通用指令。我们来考虑一个操作码为7（十六进制数）的指令，它请求将两个寄存器的内容进行OR（或）运算。（在2.4节中，我们会知道两个寄存器的“或”运算意味着什么。现在我们感兴趣的只是指令是如何译码的。）在这种情况下，下一个十六进制数字将指示存放运算结果的寄存器，而操作数最后两个十六进制数字

指示要对哪两个寄存器进行“或”运算。因此。指令70C5指的是“将寄存器C和寄存器5的内容进行‘或’运算，并将结果存入寄存器0”。

我们所讲的机器的两条LOAD指令存在细微的差别。这里，操作码1（十六进制）表示将某一存储单元内容载入寄存器的指令，操作码2（十六进制数）表示一个指令把一个特定的值加载到寄存器。它们的差别在于，第一种类型的指令操作数字段包含1个地址，第二种指令的操作数字段包含了一个要载入的实际位模式。

注意，该机器有两条ADD（加法）指令，一条用于二进制补码记数法表示的数值相加，一个用于浮点记数法表示的数值相加。把它们区分开来的原因是，这两种加法在算术/逻辑单元内部有不同的实现步骤。

我们用图2-7结束本节，图2-7把图2-2中的指令编码为机器语言。我们已经假定，相加的数值以二进制补码记数法形式存储在存储地址6C和6D中，其相加的结果存放在地址为6E的存储单元里。

指令编码	翻　译
156C	把地址为6C的存储单元里的位模式载入寄存器5
166D	把地址为6D的存储单元里的位模式载入寄存器6
5056	把寄存器5和6的内容按二进制补码表示相加，结果存入寄存器0
306E	把寄存器0的内容存放到地址为6E的存储单元中
C000	停止

图2-7　图2-2中指令的编码形式

问题与练习

1. 对于数据在计算机不同位置之间移动的操作，使用“移动”（move）这一术语为什么是用词不当？
2. 在本书中，JUMP指令是通过给出目的地名称（或步骤号）的方法来表示的（例如“跳到步骤6”）。这种方法的缺点是，如果一个指令名称（或步骤号）后来改变了，那么必须寻找所有转移到该指令的JUMP指令，并改变这些JUMP指令中的目的地。请另外设计一种表达JUMP指令的方法，使得不需要明确给出目的地的名字。
3. 指令“如果0等于0，那么就转移到步骤7”是无条件转移还是条件转移？为什么？
4. 用实际的位模式编写图2-7中的示例程序。
5. 下列指令是用附录C描述的机器语言编写的。请用自然语言解释这些指令。

   a. 368A　b. BADE　c. 803C　d. 40F4
6. 在附录C描述的机器语言里，指令15AB和25AB有什么区别？
7. 下面是用自然语言描述的一些指令。请把它们翻译为附录C中描述的机器语言。
   a. 将十六进制数值56装入（LOAD）寄存器3中。
   b. 将寄存器5循环（ROTATE）右移3位。
   c. 对寄存器A和寄存器5执行与（AND）操作，并将其结果存入寄存器0中。

2.3　程序执行

计算机总是按照需要把存储器里的指令复制到CPU中来执行存储器中的程序。一旦指令到达CPU，每个指令就会被解码及执行。从存储器中取指令的顺序与这些指令存储在存储器中的顺序相对应，除非被JUMP指令更改。

为了理解整个执行过程如何进行，我们有必要仔细了解一下CPU内部的2个专用寄存器，**指令寄存器**（instruction register）和**程序计数器**（program counter）（见图2-4）。指令寄存器用于存

储正在执行的指令；程序计数器包含下一个待执行指令的地址，因此它用于以机器方式跟踪程序执行到了什么地方。

CPU通过不断重复执行一个算法来完成工作，该算法引导它完成一个称为**机器周期**（machine cycle）的三步处理。该机器周期的3个步骤分别为取指、译码和执行（图2-8）。在取指步骤，CPU根据程序计数器指定的地址，请求主存储器给它提供存放在该地址的指令。因为我们的计算机的每一条指令长度为2个字节，所以取指过程需要从主存储器中读取2个存储单元的内容。CPU将从存储器中读取的指令存放在指令寄存器中，然后将程序计数器的值加2，使得程序计数器包含下一条要执行的指令的存储单元地址。这时，程序计数器为下一次取指做好了准备。

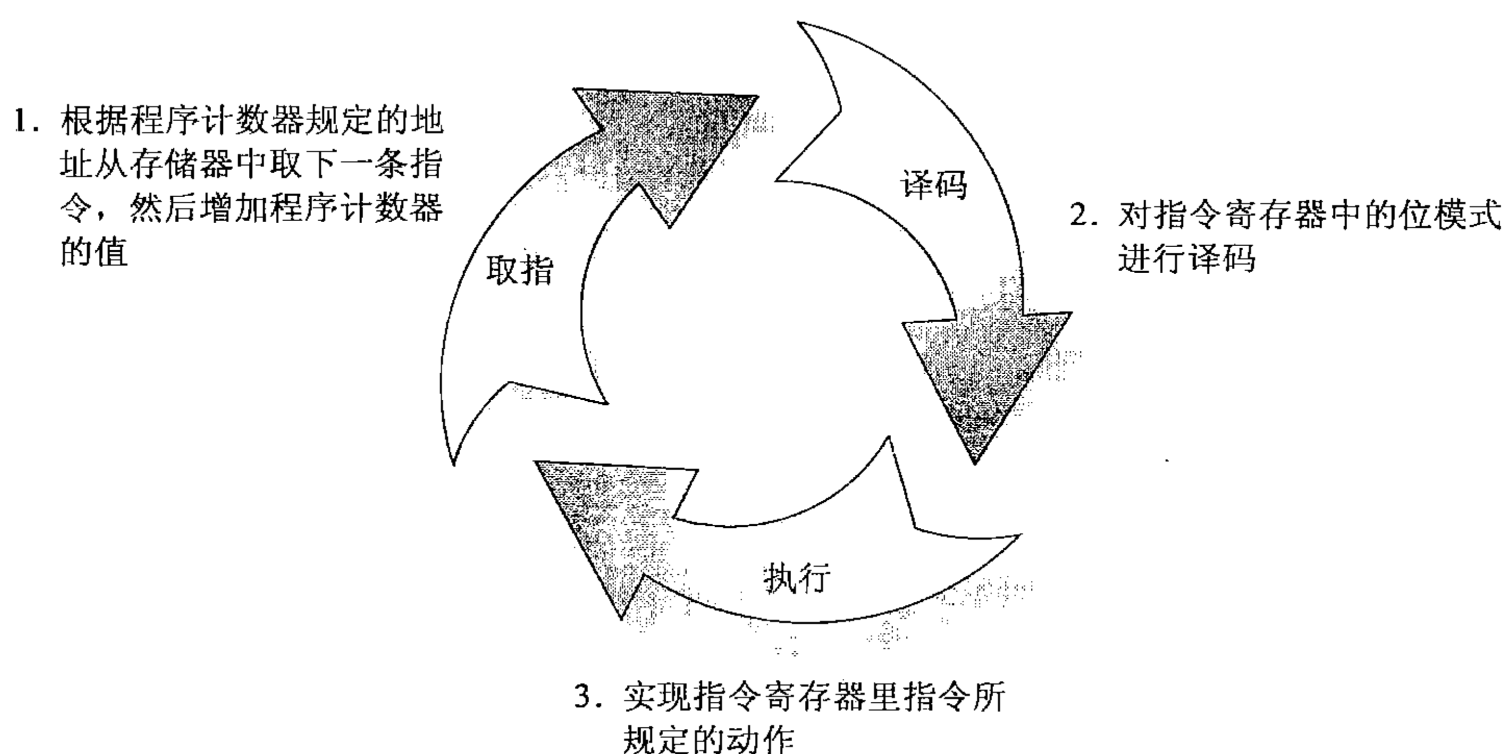


图2-8　机器周期

比较计算机能力

购买个人电脑时通常用时钟速度来比较计算机。计算机的**时钟**（clock）是一个称为振荡器的电路，生成用于协调计算机活动的脉冲——该振荡器电路生成脉冲越快，则机器周期的执行速度也越快。时钟速度以赫兹（缩写为Hz）为单位，1Hz相当于每秒1个周期（或脉冲）。台式计算机比较典型的时钟周期是几百MHz（较老的型号）到几GHz。（MHz是megahertz的缩写，1 MHz=10^6 Hz，GHz是gigahertz的缩写，1 GHz=1000 MHz。）

不同的CPU设计在一个时钟周期里完成的工作量是不同的。因此，在比较具有不同CPU的计算机时，单单比较时钟速度没有太大的意义。如果你在比较两台计算机，一台基于英特尔处理器，一台基于ARM处理器，那么采用**基准测试**（benchmark）来进行比较则更有意义。基准测试是指，在比较不同计算机时让它们执行同样的程序（称为基准），然后比较它们的性能。通过选择代表不同类型应用的基准，就能够从这些比较得到对各类市场有意义的结果。

由于指令现在已经存入了指令寄存器，CPU对该指令译码，其中包括根据该指令的操作码将操作数字段分解为适当的部分。

然后，CPU激活相应电路以执行指令，完成所请求的任务。例如，如果该指令是从存储器中加载，CPU将给主存储器发送相应信号，等待其发送数据，再将数据存入要求的寄存器；如果该指令是算术运算，CPU将激活算术/逻辑单元中相应的电路，并将正确的寄存器作为输入，等待算术/逻辑单元计算结果并将结果存入相应的寄存器。

一旦指令寄存器中的指令执行完毕，CPU又将从取指步骤开始下一个机器周期。注意，由于

程序计数器在前一个取指步骤的最后已经增加了值，所以它再次为CPU提供了正确的指令地址。

JUMP指令的执行比较特殊。例如，指令B258（见图2-9）的含义是“如果寄存器2的内容与寄存器0的内容相同，则跳转到地址为58（十六进制）的指令”。此时，该机器周期的执行步骤首先是比较寄存器2和寄存器0。如果它们包括不同的位模式，那么执行步骤结束，并开始下一个机器周期。不过，如果这两个寄存器内容相同，那么机器在执行此步骤时要将数值58（十六进制）存入程序计数器里。在这种情况下，下一个读取指令步骤发现程序计数器值为58，于是该地址的指令将为下一条被读取和执行的指令。

注意，如果该指令为B058，那么该程序计数器是否需要更改取决于寄存器0和寄存器0的内容是否相同。但是，它们是相同的寄存器，因此必然有相同的内容。于是，无论寄存器0的内容是什么，形式为B0XY的指令都将使程序跳转至存储位置XY并执行。

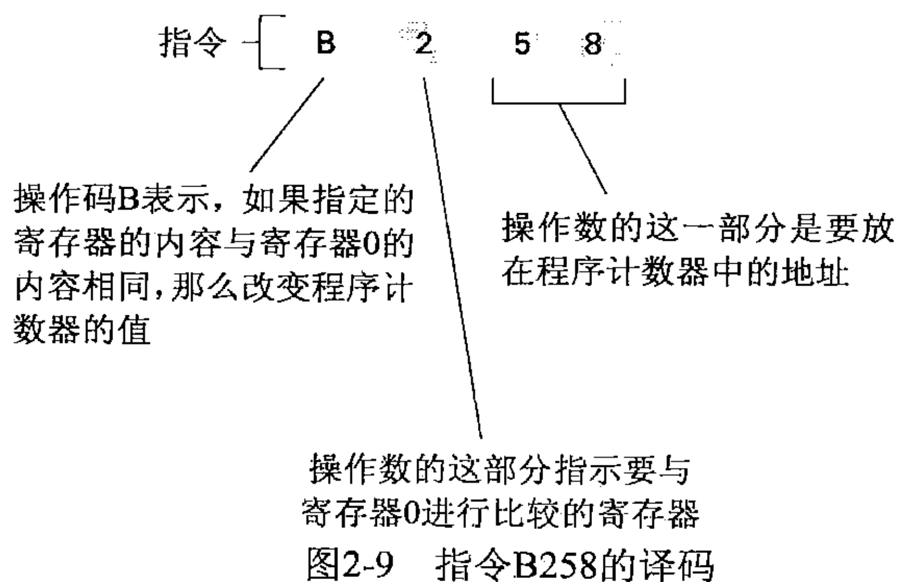


图2-9　指令B258的译码

2.3.1　程序执行的一个例子

让我们从机器周期的角度看图2-7的程序是如何执行的。该程序从主存储器中取出两个值，计算它们的和，并将结果存储在一个主存储单元里。首先，我们需要将该程序存放在存储器的某个地方。对于这个例子，假设该程序存放在从A0（十六进制）开始的连续地址中。在按照这种方法存放好该程序后，要执行这个程序只需要把该程序的第一条指令的地址（A0）存放在程序计数器中并开启机器（见图2-10）。

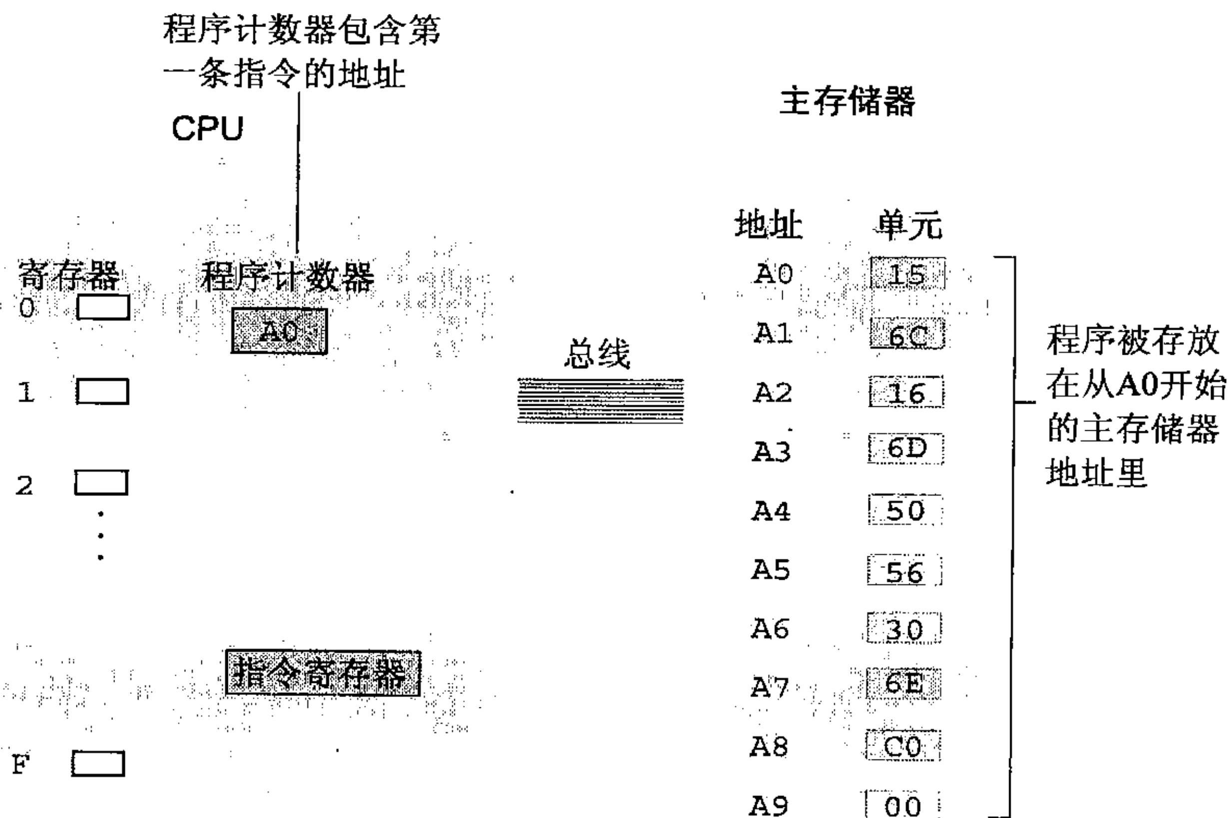
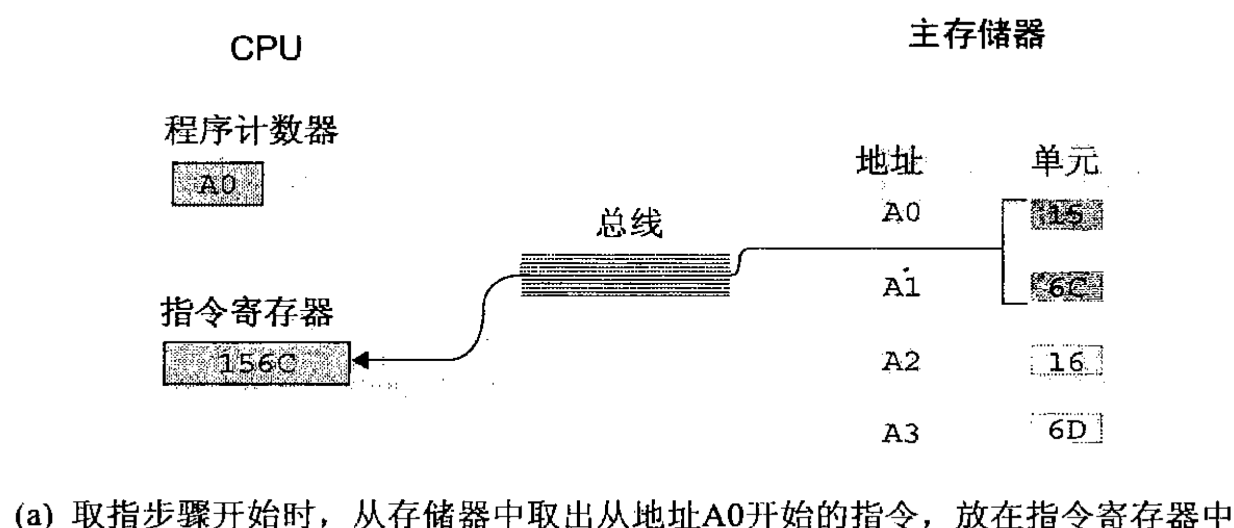


图2-10　图2-7中的程序被存储在主存储器中并准备执行

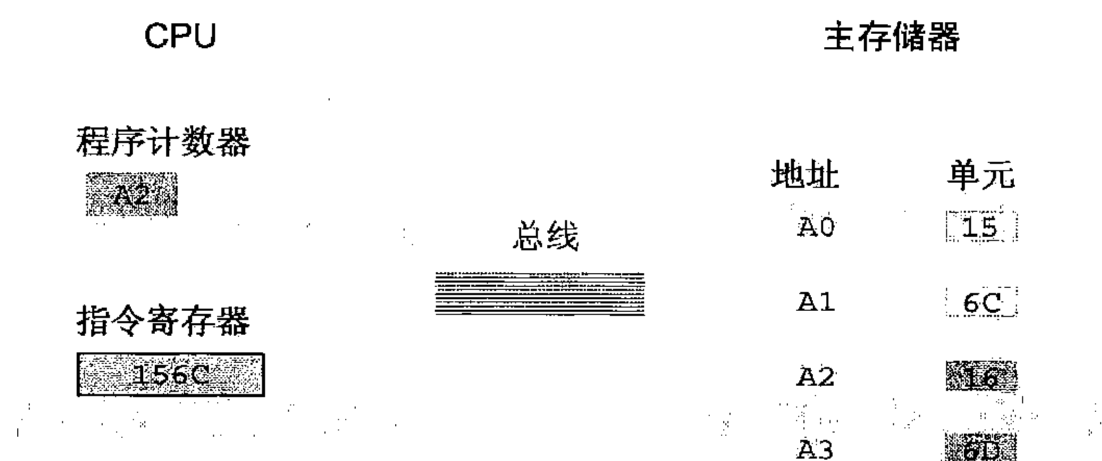
CPU开始机器周期的取指步骤，该步骤把存放在地址A0的指令从主存储器中取出，并将该指令（156C）放入指令寄存器里（图2-11a）。注意，在我们的计算机里指令的长度为16位（2个字节）。于是，要读取的整个指令占用了存储单元2个地址，即A0和A1。CPU的设计考虑到了这点，使得取指时能够读两个存储单元的内容，并将获得的位模式存放在长度为16位的指令寄存器中。接着，CPU给程序计数器加2，使得该寄存器包含下一条指令的地址（图2-11b）。在第一个机器周期的取指步骤结束时，程序计数器和指令寄存器包含下列的数据：

程序计数器：A2

指令寄存器：156C



(a) 取指步骤开始时，从存储器中取出从地址A0开始的指令，放在指令寄存器中



(b) 然后增加程序计数器的值，使它指向下一条指令

图2-11　执行机器周期的读取步骤

然后，CPU要分析指令寄存器中的指令，并得出结论：它要把地址为6C的存储单元的内容加载到寄存器5中。该加载工作是在机器周期的执行步骤完成的，接着CPU开始下一个机器周期。

这个周期首先要从以地址A2开始的2个存储单元中取指令166D。CPU要将此指令存入指令寄存器，并将程序计数器增加为A4。因此，此时的程序计数器和指令寄存器中的值如下：

程序计数器：A4

指令寄存器：166D

现在，CPU对指令166D进行译码，确定它要将地址为6D的存储单元的内容加载到寄存器6，然后执行该指令。这时候，寄存器6才真正加载了数据。

因为该程序计数器现在的值是A4，所以CPU从这个地址开始取下一条指令。于是，指令5056被放入指令寄存器，程序计数器增加为A6。现在，CPU对指令寄存器的内容进行译码，然后激活二进制补码加法电路，以寄存器5和寄存器6作为输入执行加法运算。

在该执行步骤中，算术/逻辑单元执行所请求的加法运算，将结果存入寄存器0（如控制单元所要求的那样），然后向控制单元报告它已经完成了任务。然后CPU开始另一个机器周期，再一次借助程序计数器，从以地址A6开始的2个存储单元中取下一条指令（306E），然后将程序计

数器增加到A8，接着译码并执行该指令。此时，和已经被放在了存储单元6E中。

下一条指令从存储单元A8开始读取，同时程序计数器被增加到AA。指令寄存器（C000）的内容现在被译码为停止指令。因此，机器在该机器周期的执行步骤停止，程序完成。

概括地说，如果我们遵照程序详细的指令列表，那么一个存放在存储器里的程序的执行过程就如同你我都可以做的那样。我们可以通过标记指令来确定执行到的位置，而CPU则利用程序计数器来确定执行到的位置。在确定下一步要执行什么指令后，我们需要读该指令并分析它的含义，然后实现所请求的任务并返回到指令的列表，为下一条指令做准备。同样，计算机执行指令寄存器中的指令，然后从另一个取指步骤开始继续进行。

2.3.2 程序与数据

许多程序可以同时存储在计算机的主存储器里，只要它们的地址不同。开启计算机时运行哪个程序可以通过适当地设置程序计数器来决定。

然而，我们必须记住：数据也存储在主存储器中，也用0和1来编码，所以计算机自己无法知道哪是数据，哪是程序。如果程序计数器被赋予了数据的地址而非所希望的程序的地址，那么在没有更好选择的情况下，该CPU会像取指令一样读取此数据的位模式并执行。最终的结果取决于该数据。

然而，我们不应该就此得出结论，以为将程序和数据以相同的形式存入计算机的存储器是错误的。事实上，这已经被证明是一个有用的特性，因为它使得某一个程序可以操控其他程序（甚至是自己），就像它可以操控数据一样。由此可以设想有这样的一个程序，它可以根据其与环境的交互自我修正，因此展现了它的学习能力；亦或者有这样一个程序，它可以编写及执行其他程序，以解决它所遇到的问题。

问题与练习

1. 假设在附录C描述的计算机中，从地址00到05的存储单元中包含下列（十六进制）位模式：

地　址	内　容
00	14
01	02
02	34
03	17
04	C0
05	00

如果启动计算机时，程序计数器被设为00，那么该计算机停止时，地址为17（十六进制）的存储单元里会有什么样的位模式？

2. 假设在附录C描述的计算机中，从地址B0到B8的存储单元中包含下列（十六进制）位模式：

地　址	内　容
B0	13
B1	B8
B2	A3
B3	02
B4	33
B5	B8
B6	C0
B7	00
B8	0F

a. 如果程序计数器最初为B0，执行第一条指令之后，存储在寄存器3中的位模式是什么？

b. 在执行停止指令时，存储单元B8里的位模式是什么？

3. 假设在附录C描述的计算机中，从地址A4到B1的存储单元中包含下列（十六进制）位模式：

地 址	内 容
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

假设该计算机启动时程序计数器含有值A4，回答下面的问题。

a. 地址为AA的指令第一次执行时，寄存器0中是什么？

b. 地址为AA的指令第二次执行时，寄存器0中是什么？

c. 该计算机停止之前，存放在地址AA里的指令执行了多少次？

4. 假设在附录C描述的计算机中，从地址F0~F9的存储单元中包含下列（十六进制）位模式：

地 址	内 容
F0	20
F1	C0
F2	30
F3	F8
F4	20
F5	00
F6	30
F7	F9
F8	FF
F9	FF

如果启动计算机时程序计数器含有值F0，当到达地址为F8的指令时，该计算机执行什么？

*2.4 算术/逻辑指令

如前所述，算术/逻辑指令组由算术、逻辑、移位等运算指令组成。在本节中，我们将详细介绍这些运算。

2.4.1 逻辑运算

第1章介绍了逻辑运算AND（与）、OR（或）和XOR（异或），它们都组合两个输入的二进制位，得到一个输出的二进制位。这些运算可以扩展成为这样的运算：组合两个二进制位串产生一个二进制位串输出，只需要把上述基本运算应用到每一列。例如，位模式10011010与

11001001进行与（AND）运算所得结果如下：

```
    10011010
AND 11001001
    10001000
```

这里，我们只是在每一列上将两个二进制位进行AND的结果作为结果。同理，将这些位模式进行OR运算及XOR运算时得到：

```
   10011010        10011010
OR 11001001    XOR 11001001
   11011011        01010011
```

AND运算的一个主要用途是将位模式的一部分设为0，而不影响另外一部分。例如，如果字节00001111是AND运算的第一个操作数，那么会产生什么结果？即使不知道第二个操作数的内容，我们仍然能够得出这样的结论，即结果的最高4位均为0。其次，结果的最低4位将与第二个操作数的最低4位相同，如下所示：

```
    00001111
AND 10101010
    00001010
```

AND运算的这个应用是一个称为**屏蔽**（masking）的过程的例子。这里，一个称为**掩码**（mask）的操作数决定另一个操作数的哪个部分会影响结果。在这个AND运算中，屏蔽得出的结果中，其中一部分是一个操作数的复制品，而没有复制的部分为0。

此类运算在操作一个**位图**（bit map）时很实用。位图是由若干二进制位组成的串，其中每个位表示一个特定对象存在与否。在讨论图像表示的时候已经涉及位图，那时每一位是与一个像素相联系的。再举一例，一个由52位组成的串，其中每个位与一张特定的扑克牌相联系，那么此位串可以表示一手5张牌，我们只需要将1赋予和手中牌相对应的5个位，而将0赋予其他位。同理，13个位为1的52位位图可以表示一手桥牌，32位位图可以表示32种口味冰激凌的有无。

接着，假设一个8位存储单元被用作一个位图，我们希望查明与从高位位算起的第3位相联系的对象是否存在？我们只需要将整个字节与掩码00100000进行AND操作，当且仅当该位图从高位端算起的第3位本身为0时，结果的字节值全为0。于是，在该AND运算中安排一个条件转移指令，程序就可以实现相应的动作。其次，如果该位图从高位算起的第3位为1，我们想在不破坏其他位的情况下将其改为0，那么可以把该位图与掩码11011111进行AND操作，然后用结果替代原来的位图。

AND运算可用于复制一个位串的一部分，方法是在不复制的部分置0，而OR运算也可用于复制一个串的一部分，但在不复制的部分置1。为此，我们再次使用掩码，但是这次用0来指示要复制的位的位置，用1指示不复制的位置。例如，将任何字节和11110000进行OR运算，结果的位模式的最高有效4位为1，而剩余位则是另外一个操作数最低有效4位的副本，如下所示：

```
   11110000
OR 10101010
   11111010
```

因此，掩码11011111参与AND运算一定可以使得一个8位位图最高起第3位变为0，而掩码00100000可以参与OR运算将同样的位置变为1。

XOR运算的一个主要用途是形成一个位串的反码。将任意一个字节与全部为1的掩码进行XOR运算可以得到该字节的反码。例如，注意下面示例中第二个操作数与其结果的关系：

```
    11111111
XOR 10101010
    01010101
```

在附录C描述的计算机语言中，操作码7、8和9分别用于逻辑OR、AND和XOR。每个操作码要求在指定的2个寄存器内容之间进行相应的逻辑运算，并将结果存放在另一个指定的寄存器中。例如，指令7ABC要求的是：将寄存器B和C的内容进行OR运算，并将结果存放在寄存器A中。

2.4.2 循环移位及移位运算

循环及移位运算用于移动寄存器内的二进制位，常用于解决对齐问题。这些运算根据移动方向（向右或向左）和其过程是否循环来分类。这些分类准则的混合使用产生了许许多多变体。下面我们简单介绍一下所涉及的概念。

我们来考虑含一个字节二进制位的寄存器。如果我们将其内容向右移一位，则最右边的位落到了边界以外，最左端出现了一个空位。对于这个移出的位及留出的空位如何操作是区别各种移位运算的特征。一种方法就是将右侧移出的位放置在左端的空位上，这类运算称为**循环移位**（circular shift或rotation）。因此，如果我们针对一个字节的位模式向右进行8次循环移位，则所得位模式与初始形式相同。

另外一种方法就是丢弃移出边界的位，并用0填充空位，这类运算称为**逻辑移位**（logical shift）。这种向左的移位可以实现2乘以二进制补码的运算。因为，二进制数字左移相当于乘2，而对一个十进制数字左移就相当于乘10。此外，除2运算可以通过二进制位右移来完成。无论对于哪种移位，在使用某种记数法系统时都一定要小心地保留符号位。因此，右移时留出的空位（符号位位置）总是用它原来的值来填。保留符号位不变的移位称为**算术移位**（arithmetic shift）。

在各种可能的移位和循环移位指令中，附录C描述的机器语言仅包括一条右循环移位指令，操作码为A。在这个移位运算中，操作数的第一个十六进制数字指定了要循环移位的寄存器，操作数的其余部位规定要循环移位的位数。因此，指令A501意为“将寄存器5的内容循环右移1位”。特别地，如果寄存器5最初包含位模式65（十六进制），那么执行此指令（见图2-12）后将包含B2。（尝试一下如何组合使用附录C描述的机器语言提供的指令来产生其他移位及循环移位指令。例如，因为一个寄存器的长度为8位，所以循环右移3位与循环左移5位所得结果一样。）

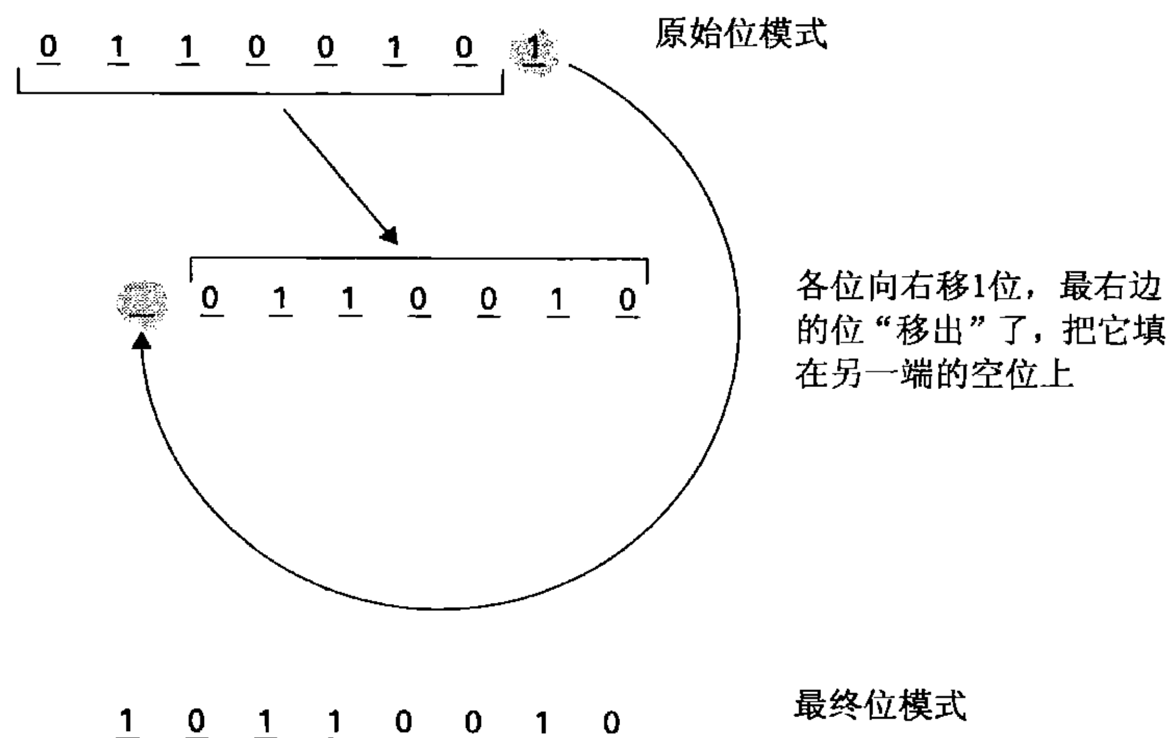


图2-12 将位模式65（十六进制）循环右移1位

2.4.3 算术运算

尽管我们已经提到过算术运算——加、减、乘和除，但还需饶舌几句。首先，我们知道减法运算可以通过加法及取负来模拟。此外，乘法只不过就是反复进行加法运算，除法就是反复进行减法运算。（6可以减去3个2，所以6除以2为3。）因此，一些小型CPU都只装有加法指令或者加法和减法指令。

我们还应该注意到，每种算术运算都有许多变体。对于附录C描述的机器语言里使用的加法运算，我们已经暗示了这一点。例如，对于加法运算，如果相加的数值用二进制补码记数法存储，此加法过程的实现就一定是每列数字直接相加。然而，如果操作数用浮点记数法存储，加法过程则为读取每个操作数的尾数，根据指数字段将其向左或右移位，检查符号位，执行加法，最后将其结果翻译成浮点记数法。因此，尽管它们都是加法运算，但是计算机的实现过程并不相同。

问题与练习

1. 完成指定的运算。

```
a.      01001011    b.      10000011    c.      11111111
    AND 10101011        AND 11101100        AND 00101101

d.      01001011    e.      10000011    f.      11111111
    OR  10101011        OR  11101100        OR  00101101

g.      01001011    h.      10000011    i.      11111111
    XOR 10101011        XOR 11101100        XOR 00101101
```

2. 假如想从一个字节中分离出中间的4位：将其他4个位设为0，却不干扰中间的4位。请问必须使用什么掩码及什么运算？
3. 假如想对一个字节的中间4位取反码，其他4位保持不变。请问必须使用什么掩码及什么运算？
4. a. 假如想对一个位串的前2位进行XOR运算，然后以下列方式继续下去：把上次计算的结果与位串的下一个位进行XOR运算。请问最后的计算结果与该串中1的个数有什么关系？
   b. 在对一个报文编码时，需要确定使用什么样的奇偶校验位，问题a与此问题有什么联系？
5. 通常使用逻辑运算代替数值运算是很方便的。例如，逻辑运算AND将两个位组合的方法同乘法运算一样。哪一种逻辑运算和两个位的加法几乎相同，这样情况下会导致什么问题？
6. 将ASCII码中的小写字母变为大写字母，请问需要使用什么逻辑运算和什么掩码？大写字母变小写时又如何？
7. 在下列位串中，完成循环右移3位会得到什么结果？
   a. 01101010　　b. 00001111　　c. 01111111
8. 对于下面用十六进制表示的字节，执行循环左移1位的运算，结果是什么？请用十六进制形式给出答案。
   a. AB　　b. 5C　　c. B7　　d. 35
9. 一个8位位串循环右移3位等价于循环左移多少位？
10. 如果位模式01101010与11001100是以二进制补码记数法表示的值，请问它们的和的位模式是什么？如果这两个位模式是用第1章讨论的浮点格式表示的，那么结果又是什么？
11. 使用附录C描述的机器语言编写一个程序：它将地址为A7的存储单元的最高有效位设为1，但不影响其他位的值。
12. 使用附录C描述的机器语言编写一个程序：它将存储单元E0的中间4位复制到存储单元E1中的最低4位，并将存储单元E1的最高4位置为0。

*2.5 与其他设备通信

主存储器和CPU构成了计算机的核心。本节将研究这个核心（将称它为计算机）如何与外围设备通信，如海量存储系统、打印机、键盘、鼠标、监视器、数码相机以及其他计算机。

2.5.1 控制器的作用

计算机与其他设备的通信通常是通过称为**控制器**（controller）的中间设备来处理的。对于个人计算机，控制器可能由永久安装在其主板上的电路组成，或者为了灵活，它会采用电路板的形式插入主板的插槽中。无论哪种形式，控制器都是通过电缆与计算机箱里的外围设备相连接的，或者与计算机背面称为**端口**（port）的连接器相连接，其他外围设备可以插到这些端口上。这些控制器有时候本身就是小型计算机，每个都有自己的存储电路和简单的CPU，可以实现指挥该控制器活动的程序。

控制器将信息和数据来回地在两种形式之间转换：一种是与计算机内部特征相适应的形式，另外一种是与所连接外围设备相符的形式。最初，每个控制器都是为特定类型的设备设计的。因此，购买一种新的外围设备常常也就需要同时购买一个新控制器。

最近，人们已经开始在个人电脑领域开发标准，例如USB（Universal Serial Bus，通用串行总线）和FireWire（火线），这样一个控制器就可以处理多种设备。例如，一个USB控制器可以用作计算机与其他任何同USB兼容的系列设备的接口。现在，市场上可以与USB控制器通信的设备包括鼠标、打印机、扫描仪、海量存储设备、数码相机，以及智能手机。

每一个控制器通过连接到相同的总线（该总线用来连接计算机的CPU和主存）完成与计算机的通信（图2-13）。由于这种连接，每个控制器都能够监控CPU与主存储器之间正在发送的信号，也可以将自己的信号插入总线。

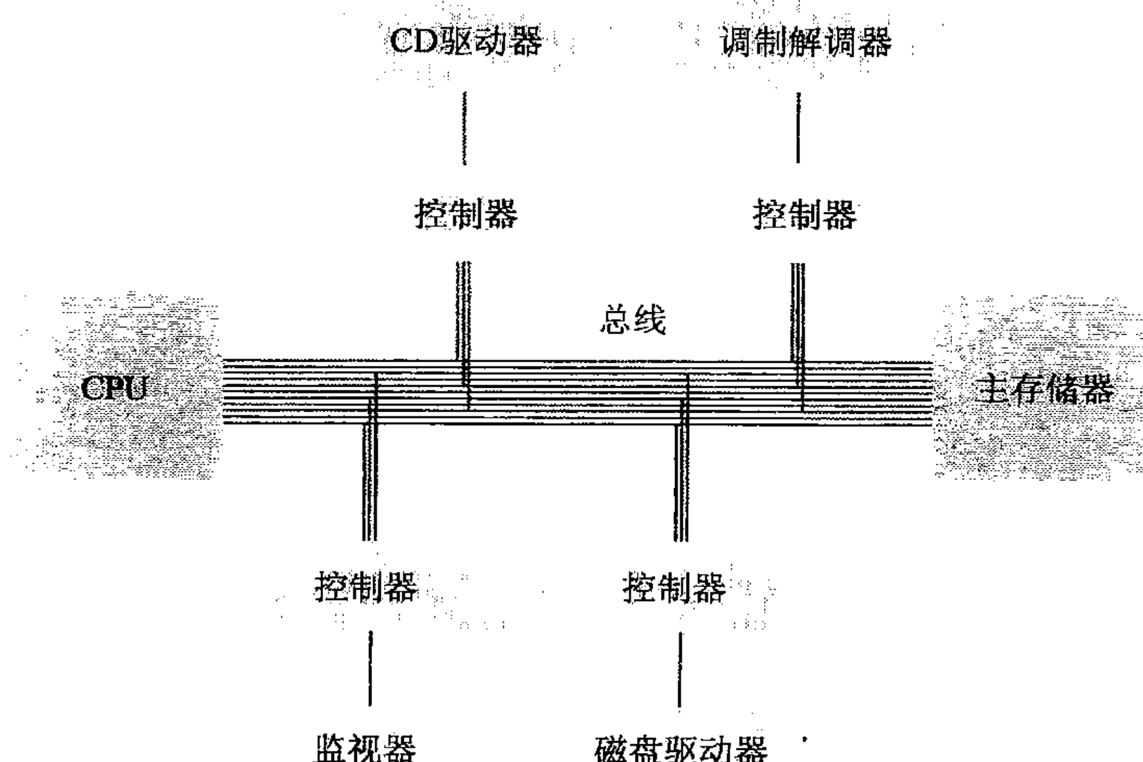


图2-13 连接到计算机总线的控制器

通过这种安排，CPU能够以与主存储器通信的方式与连接在总线上的控制器通信。为了发送一个位模式给控制器，该位模式首先要在CPU的一个通用寄存器中构建，然后由CPU执行一个类似STORE指令的指令，将该位模式“存储”到控制器中。类似地，当要从一个控制器接收一个位模式时，要使用一条类似LOAD指令的指令。

USB与FireWire

USB（通用串行总线）和FireWire（火线）是标准化的串行通信系统，它们简化了给个人电脑添加外围设备的过程。USB由英特尔公司主导研发，FireWire则由苹果公司主导研发。两者的目的都是通过一个控制器提供外部端口，并用该端口来连接各种外围设备。在该设置中，控制器将计算机内部信号特征转换成相应的USB或FireWire标准信号。反之，为了与控制器通信，与控制器相连接的每个设备都将其内部特性转换成相同的USB或FireWire标准。于是，给PC添加新设备就不再需要增加新的控制器，只需要在USB端口或FireWire端口插入分别与其兼容的设备。

对比而言，FireWire的传输速率更高，但是USB技术成本低，因此在低成本大众市场领域占据突出地位。现在，市场上兼容USB的设备有鼠标、键盘、打印机、扫描仪、数码相机、智能手机，以及为备份应用设计的海量存储系统。FireWire应用趋向于集中在需要更高传输速率的设备上，例如摄像机和联机海量存储系统。

在某些计算机的设计中，通过控制器的数据传输（输入与输出）直接使用LOAD和STORE操作码（虽然这些操作码已经用于同主存储器的通信）。在这种情况下，每个控制器都被设计为响应唯一一组地址的引用，而主存储器被设计成忽略对这些地址的引用。因此，当CPU在总线上发送一条消息，要把一个位模式存储到一个分配给某个控制器的存储器地址时，这个位模式实际上是存储到该控制器中，而不是主存储器中。同理，如果CPU试图从这样的存储器地址接收数据（如LOAD指令），那么它所接收到的位模式将来自控制器而不是存储器。这样的通信系统称为**存储映射输入/输出**（memory-mapped I/O），因为该计算机的输入/输出设备好像是在各种存储器位置里（图2-14）。

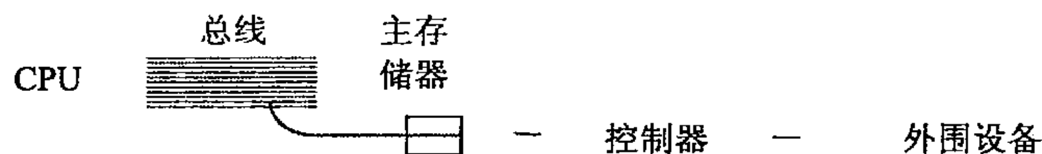


图2-14 存储映射输入/输出的概念表示

另外一种存储器映射输入/输出的方法是在机器语言中提供特定的操作码，用以规定通过控制器的数据传输（输入与输出）。具有这些操作码的指令称为I/O指令。例如，如果附录C中描述的机器语言遵循这种方法，它可能包括诸如F5A3这样的一条指令，F5A3指的是“将寄存器5的内容存储在由位模式A3指定的控制器中”。

2.5.2 直接内存存取

因为控制器是连接到一台计算机的总线上的，所以它就有可能在CPU不使用总线的几纳秒时间里实现与主存储器的通信。控制器这种存取主存储器的能力称为DMA（Direct Memory Access，直接存储器存取），可以极大地提高计算机的性能。例如，要从磁盘的一个扇区读取数据，CPU可以将编码为位模式的请求发送给连接这个磁盘的控制器，要求该控制器读取这个扇区并将数据存储在指定的一块主存储器区域中。在该控制器执行此读操作并通过DMA将数据存储在主存储器时，CPU可以继续执行其他任务。于是，这两个活动会被同时执行。CPU将执行某个程序，而控制器则监视磁盘与主存储器之间的数据传输。这样，在相对缓慢的数据传输过程中，CPU的计算资源就不会被浪费。

使用DMA同样也有不利影响，它使计算机总线的通信复杂化。位模式必须在CPU与主存储

器之间、CPU与每个控制器之间，以及每个控制器与主存储器之间进行传送。协调总线上的所有这些活动是个很大的设计难题。即使设计非常出色，CPU与控制器竞争总线存取时，中央总线也可能成为障碍。此障碍称为**冯·诺依曼瓶颈**（von Neumann bottleneck），因为它源于**冯·诺依曼体系结构**（von Neumann architecture），在该结构中，CPU是通过中央总线从主存储器取指的。

2.5.3 握手

两个计算机部件之间的数据传输很少是单向进行的。即使我们可以把打印机看做是接收数据的设备，但事实上它也向计算机发送数据。毕竟，计算机产生字符并向打印机发送字符的速度要远远快于打印机能够打印的速度。如果计算机盲目地把数据发送给打印机，那么打印机很快就落在后面了，结果是使数据丢失。因此，诸如打印文件这样的过程都会包括持续的双向对话，计算机和外围设备之间交换设备状态的信息，协调它们之间的活动。这个过程称为**握手**（handshaking）。

握手通常涉及一个**状态字**（status word），它是由外围设备生成并发送给控制器的一个位模式。该状态字是一个位图，其中的各个二进制位反映了该设备的各种状态。以打印机为例，其状态字的最低有效位数值可以表示该打印机是否缺纸，而下一个位可以表示该打印机是否已经准备好再接收数据，另外还有一位可用于指出是否卡纸。控制器是自己响应这些状态信息，还是交由CPU来处理，这取决于不同的系统。无论哪种情况，状态字都提供了一种机制，用于协调与外围设备的通信。

2.5.4 流行的通信媒介

计算设备之间的通信由两种途径处理：并行及串行。这些术语指的是传输信号的方式。**并行通信**（parallel communication）指的是若干信号同时传输，每个信号都在各自的“线路”上。这种技术数据传输快，但是需要相对复杂的通信通路。例如计算机内部总线，其中多条线路被用于同时传输大量数据块及其他信号。

与此相反，**串行通信**（serial communication）指在一条信号线上一个信号接一个信号地传输。相对于并行通信，串行通信只需要一条相对简单的数据路径，这也是它很流行的原因。USB与FireWire在短短几米的距离内提供相对高速的传输速率，属于串行通信。对于相对较长的距离（在家中或者办公楼里），通过以太网连接（见4.1节）的串行通信，无论是通过电线还是无线电广播连接，都很流行。

多年来，传统的语音电话线在远距离通信方面一直主宰着个人电脑领域。这些通信路径都只有一根电线并通过它逐一传输语音信号，本质上属于串行系统。这样的数字数据传输实现过程如下：首先利用**调制解调器**（modulator-demodulator，缩写为modem）将位模式转换为听得见的音调，并通过电话系统串行传输，然后在目的地由另一个调制解调器将这些音调重新转换成二进制位。

为了通过传统的电话线实现更加快速的远距离通信，电话公司提供了一种称为DSL（Digital Subscriber Line，数字用户线路）的服务，它利用以下事实：现有的电话线实际上能够处理比传统话音通信所用频段更宽的频率范围。确切地说，DSL使用高于可听范围的频率传输数字数据，将较低频谱用于语音通信。虽然DSL已非常成功，但电话公司正迅速升级自己的系统来使用光纤线路（光纤线路比传统电话线路更容易支持数字通信）。

其他可与DSL和光纤相竞争的技术包括用于有线电视系统的电缆以及通过高频无线电广播的卫星链路。

2.5.5　通信速率

一个计算部件与另外一个计算部件之间传输数据位的速率是以bit/s（bit per second，比特/秒）计量的。常用的单位有Kbit/s（kilo-bit/s，等于10^3 bit/s）、Mbit/s（mega-bps，等于10^6 bit/s）和G bit/s（giga-bps，等于10^9 bit/s）。（注意位与字节之间的区别：8 Kbit/s相当于1 KB/s）。使用缩写形式时，小写的英文字母b通常表示位（bit），而大写的英文字母B通常表示字节（byte）。

对于短距离通信，USB及FireWire可以提供几百Mbit/s的传输速率，对于大多数多媒体应用已经足够了。再加上便利性及相对低价，如今它们已被广泛用于家用计算机与本地外围设备的通信，如与打印机、外部硬盘驱动器以及相机的通信。

通过结合**多路复用技术**（multiplexing，数据编码或混合，使得一条通信路径可完成多条通信路径的功能）及数据压缩技术，传统的语音电话系统能够支持57.6 Kbit/s的传输速率，这无法满足当今多媒体和因特网应用（如YouTube和Facebook）的需要。播放MP3音乐需要大约64 Kbit/s的传输速率，即使播放低品质的视频也需要用Mbit/s计量的传输速率。这正是能够提供Mbit/s范围的传输速率的DSL、电缆以及卫星链路等取代传统音频电话系统的原因。（例如，DSL提供的传输率大约为54 Mbit/s。）

一个特定设置可获得的最大速率，取决于通信路径的种类以及实现过程中使用的技术。这个最大速率通常大致等同于通信路径的**带宽**（bandwidth），但该术语除了传输速率还有容量的含义。也就是说，说一条通信路径具有高带宽（或提供**宽带**服务）意味着一条通信路径能以高速率传输位，同时意味着该通信路径还能够同时携带大量信息。

问题与练习

1. 假设附录C描述的机器使用存储映射输入/输出，地址B5是打印机端口所在的位置，要打印的数据应该发送给它。
   a. 如果寄存器7包含字母A的ASCII码，那么要通过打印机打印该字母，应该使用哪条机器语言指令？
   b. 如果该计算机每秒执行一百万条指令，那么在一秒钟内可以向打印机发送这个字符多少次？
   c. 如果该打印机一分钟可以打印5页传统文本，那么在(b)的情况下，它能够跟得上发送给它的字符吗？
2. 假设你的个人电脑硬盘每分钟3000转，每个磁道有16个扇区，每个扇区有1024个字节。如果磁盘控制器打算从磁盘驱动器中接收从旋转磁盘中读到的位，那么磁盘驱动器与磁盘控制器之间的通信速率大约是多少？
3. 一本以Unicode编码的300页小说，按照54 Mbit/s的速率需要传输多久？

*2.6　其他体系结构

为了拓宽视角，我们来考虑一些已经讨论过的传统计算机体系结构的替代方案。

2.6.1　流水线

电子脉冲在电线上的传播要比光速慢。光大约每纳秒（ns，十亿分之一秒）能传播1英尺的距离，然而CPU中的控制单元至少需要2 ns才能从1英尺之外的存储单元中读取到指令。（必须发送读请求到存储器，这至少需要1ns，而指令又必须送回CPU，这至少也需要1ns。）因此，在这样的机器中取指和执行一条指令需要若干纳秒——这就意味着提高计算机执行速度的问题最终将变成小型化问题。

然而，提高执行速度并不是改进计算机性能的唯一途径。真正目的是改进机器的**吞吐量**（throughput）——机器在给定时间内可以完成的工作总量。

在不要求提高执行速度的前提下，增加计算机吞吐量的一个例子涉及**流水线技术**（pipelining），该技术允许一个机器周期内各步骤重叠进行。特别是，当执行一条指令时，可以取下一条指令，这也就意味着在任何一个时刻可以有不止一条指令在“流水线”上，每条指令处在不同的处理阶段。这样，尽管读取和执行每条指令的时间保持不变，计算机的总吞吐量却提高了。（当然，当到达一条JUMP（转移）指令时，不会实现预取指令来提高效率的效果，因为“流水线”上的指令不是所需要的。）

现代计算机设计已使得流水线思想大大超越了我们所举的例子。它们经常能够同时读取若干条指令，并且一次可以执行多条彼此互不依赖的指令。

多核CPU

科技进步使得可以放置越来越多的电路在一个硅片上，以致计算机部件之间的物理差别逐渐变小。例如，单个芯片就可以包括CPU和主存储器。这是片上系统方法的一个例子，目的是在单个设备中提供一个完整的系统，使得其可在更高的设计层面被用作一个抽象工具。在其他情况下，单个设备中还包含相同电路的多个复制品。它最初的形式是包含若干独立门或者多重触发器的芯片。在今天的技术条件下，单个芯片可以存放不止一个完整的CPU。这就是多核CPU设备的基础体系结构：在同一芯片上存在两个或更多CPU以及共用的高速缓冲存储器。（包含两个处理单元的多核CPU通常被称作双核CPU。）这种设备简化了MIMD系统的构建，并已被迅速应用于家用计算机。

2.6.2 多处理器计算机

可以把流水线技术看做迈向**并行处理技术**（parallel processing）的第一步，并行处理技术是若干活动在同一时间里实现的性能。然而，真正的并行处理技术需要多个处理单元，于是产生了多处理器计算机。

当今许多计算机的设计都基于这种思想，其中一个策略是将若干处理单元都连接到同一个主存储器上，其中每一个都像单处理器机器中的CPU。在这样的配置下，各处理器可以独立地工作，并通过把相关的信息放在公共存储单元里来协调各自的工作。例如，当某个处理器遇到一个大任务时，它可以将部分任务的程序存储在这个公共存储器中，然后请求其他处理器执行它。结果产生这样的计算机：不同的指令序列在不同的数据集上操作，相对于较传统的SISD（Single-Instruction stream，Single-Data stream，单指令流单数据流）体系结构，它称为MIMD（Multiple-Instruction stream，Multiple-Data stream，多指令流多数据流）体系结构。

多处理器体系结构的一个变体是将多个处理器链接起来，使得它们一起执行同一个指令序列，每个处理器都有各自的数据集。这就产生了SIMD（Single-Instruction stream, Multiple-Data stream，单指令流多数据流）体系结构。这种计算机适用于这样的应用：在一大堆数据中，对于其中每组类似的数据项都要执行同样的任务。

并行处理的另外一种方法是将许多小型机器聚集成为大的计算机，每台机器都有自己的存储器和CPU。这样，每台小型机器都与它相邻的一台或几台机器相连接，使得赋予整个系统的任务可以分割到各台小机器上实现。因此，如果分配给某台内部机器的一项任务可以分割为若干独立的子任务，那么这台内部机器可以请求它的邻居们并发地完成各个子任务。这样，完成任务的时间可以比由一台单处理器计算机独立完成任务所需要的时间少得多。

问题与练习

1. 回到2.3节问题3，如果该计算机使用了本书讨论的流水线技术，当地址为AA的指令被执行时，“流水线”里有什么？在什么条件下，该程序不会从流水线技术得到好处？
2. 在一台流水线计算机上运行2.3节问题4中的程序时，必须要解决什么冲突？
3. 假设有两个“中央”处理器连接到同一个存储器上，但执行不同的程序；再假设，其中一个处理器需要给一个存储单元的内容加1，几乎同时，另外一个处理器需要给同一个存储单元的内容减1。（净效果应该是该存储单元最终保持开始时的值。）
   a. 描述一个执行序列，其结果为该单元最终数值比开始值少1。
   b. 描述一个执行序列，其结果为该单元最终数值比开始值大1。

复习题

（带*的题目涉及选读小节的内容。）

1. a. 在什么情况下，通用寄存器和主存储单元类似？
   b. 在什么情况下，通用寄存器和主存储单元不同？
2. 根据附录C描述的机器语言回答下列问题。
   a. 将指令2304（十六进制）写成16位位串。
   b. 将指令B2A5（十六进制）的操作码写成4位位串。
   c. 将指令B2A5（十六进制）的操作数字段写成12位位串。
3. 假设在附录C描述的机器里一个数据块存储在地址从98到A2（含）的存储单元中。这一数据块占据多少存储单元？列出它们的地址。
4. 在附录C描述的机器里，刚执行完指令B0CD后程序计数器的值是多少？
5. 假设在附录C描述的机器里，从地址00到05的存储单元中包含下列位模式。

地　址	内　容
00	22
01	11
02	32
03	02
04	C0
05	00

   假定该程序计数器初始值为00，请记录该程序执行到停止这一过程中在每个机器周期取指阶段末尾，程序计数器、指令寄存器以及地址为02的存储单元的内容。
6. 假设3个值x、y、z存储在机器的存储器中。描述在计算$x+y+z$时发生的事件序列（如从存储器装入寄存器，将值保存在存储器，等等）。计算$(2x)+y$时又如何呢？
7. 下面是用附录C描述的机器语言编写的指令。把它翻译成为自然语言。
   a. 7123　b. 40E1　c. A304
   d. B100　e. 2BCD
8. 假设有一机器语言，指令的操作码字段为4位。那么该语言可以有多少条不同的指令？如果操作码字段增加到8位呢？
9. 将下列指令由自然语言翻译为附录C描述的机器语言。
   a. 将十六进制值77装入（LOAD）寄存器6。
   b. 将存储单元77的内容装入寄存器7。
   c. 如果寄存器0的内容与寄存器A的值相同，则转移（JUMP）到存储器地址为24的指令。
   d. 将寄存器4循环右移（ROTATE）4位。
   e. 将寄存器E和2的内容进行AND运算，结果存于寄存器1。
10. 重写图2-7中的程序，假定相加的数值用浮点记数法编码，而不是二进制补码记数法。
11. 下面是用附录C描述的机器语言编写的指令。请按照它们的执行是否改变存储地址为3B的存储单元的值、是否读取地址为3C的存储单元的内容、是否与地址为3C的存储单元的内容没有关系进行分类。
   a. 353C　b. 253C　c. 153C
   d. 3C3C　e. 403C
12. 假设附录C描述的机器里，从地址00到03的存储单元中包含下列位模式。

地址	内容
00	26
01	55
02	C0
03	00

a. 将第一条指令翻译为自然语言。

b. 如果机器在程序计数器的值为00时启动，那么机器停止时寄存器6中是什么位模式？

13. 假设附录C描述的机器里，从地址00到02的存储单元中包含下列位模式。

地址	内容
00	12
01	21
02	34

a. 如果机器在程序计数器值为00时启动，执行的第一条指令会是什么？

b. 如果机器在程序计数器值为01时启动，执行的第一条指令会是什么？

14. 假设在附录C描述的机器里，从地址00到05的存储单元中包含下面的位模式。

地址	内容
00	12
01	02
02	32
03	42
04	C0
05	00

假设机器在程序计数器的值为00时启动，回答下面的问题。

a. 将要执行的指令翻译成自然语言。

b. 当机器停止时，地址为42的存储单元中是什么位模式？

c. 当机器停止时，程序计数器中是什么位模式？

15. 假设在附录C描述的机器里，从地址00到09的存储单元中包含下列位模式。

地址	内容
00	1C
01	03
02	2B
03	03
04	5A
05	BC
06	3A
07	00
08	C0
09	00

假设机器在程序计数器的值为00时启动，回答下列问题。

a. 当机器停止时，地址为00的存储单元里有什么？

b. 当机器停止时，程序计数器中会是什么位模式？

16. 假设在附录C描述的机器里，从地址00到07的存储单元中包含下列位模式。

地址	内容
00	2B
01	07
02	3B
03	06
04	C0
05	00
06	00
07	23

a. 假设机器在程序计数器的值为00时启动，请列出包含待执行程序的存储单元的地址。

b. 列出用于存储数据的存储单元的地址。

17. 假设在附录C描述的机器里，从地址00到0D的存储单元中包含下列位模式。

地址	内容
00	20
01	04
02	21
03	01
04	40
05	12
06	51
07	12
08	B1
09	0C
0A	B0
0B	06
0C	C0
0D	00

假设机器在程序计数器的值为00时启动。

a. 当机器停止时，寄存器0中是什么位模式？
b. 当机器停止时，寄存器1中是什么位模式？
c. 当机器停止时，程序计数器中是什么位模式？

18. 假设在附录C描述的机器里，从地址F0到FD的存储单元中包含下列（十六进制）位模式。

地址	内容
F0	20
F1	00
F2	22
F3	01
F4	23
F5	04
F6	B3
F7	FC
F8	50
F9	02
FA	B0
FB	F6
FC	C0
FD	00

如果机器在程序计数器的值为F0时启动，那么当计算机最终执行到地址为FC的停机指令时，寄存器0中的值是什么？

19. 如果在附录C描述的机器每微秒（百万分之一秒）执行一条指令，那么完成问题18中的程序需用时多少？

20. 假设在附录C描述的机器里，从地址20~28的存储单元中包含下列位模式。

地址	内容
20	12
21	20
22	32
23	30
24	B0
25	21
26	24
27	C0
28	00

假设机器在程序计数器的值为20时启动。
a. 当机器停止时，寄存器0、1和2中是什么位模式？
b. 当机器停止时，地址为30的存储单元中是什么位模式？
c. 当机器停止时，地址为B0的存储单元中是什么位模式？

21. 假设在附录C描述的机器里，从地址AF到B1的存储单元中包含下列位模式。

地址	内容
AF	B0
B0	B0
B1	AF

如果机器在程序计数器的值为AF时启动，那么会发生什么？

22. 假设在附录C描述的机器里，从地址00到05的存储单元中包含下列（十六进制）位模式。

地址	内容
00	25
01	B0
02	35
03	04
04	C0
05	00

如果机器在程序计数器的值为00时启动，那么机器在什么时候会停止？

23. 对于下面每种情况，用附录C描述的机器语言编写一个小程序来完成以下任务。假定每个程序都放在从地址00开始的存储器里。
a. 将存储单元D8的值移动到存储单元B3。
b. 交换存储单元D8和B3中的值。
c. 如果存储单元44的值是00，则将值01存放在存储单元46中；否则，将值FF存放在存储单元46中。

24. 在计算机爱好者中曾经流行一种叫磁芯大战（Core Wars）的游戏——战舰游戏的变体。（术语*磁芯*来源于早期的一种存储技术，它用磁材料的小环的磁场方向表示0和1。小环称为磁芯。）这个游戏是在两个对立的程序之间玩的，每个程序分别存储在同一台计算机的存储器的不同位置里。假设该计算机轮流执行这两个程序，先执行一个程序的一条指令，再执行另一个程序的一条指令。每个程序的目标是通过把额外数据写到另外一个程序上来破坏对立程序；不过，哪个程序都不知道对方的位置。
a. 用附录C描述的机器语言编写一个程序，采用防卫的方式，以最小的代价玩此游戏。

b. 用附录C描述的机器语言编写一个程序，通过不断变动自己的位置来避免受到对立程序的袭击。更确切地说，在位置00开始编写程序，使其把自己复制到位置70，再转移到这个新位置。

c. 扩展(b)中的程序，继续将新位置的程序重新定位。具体来说，先将程序移至位置70，然后移到E0（=70+70），再移到60（=70+70+70），等等。

25. 用附录C描述的机器语言编写一个程序，用它计算存放在存储单元A0、A1、A2、A3中的浮点数的和，并将结果存入存储单元A4中。

26. 假设在附录C描述的机器里，从地址00到05的存储单元中包含下列（十六进制）位模式。

地　址	内　容
00	20
01	C0
02	30
03	04
04	00
05	00

如果机器在程序计数器的值为00时启动，会发生什么？

27. 假设在附录C描述的机器里，地址为08和09的存储单元中分别包含位模式B0和08，并且机器启动时程序计数器中包含值08，那么会发生什么？

28. 假设下列用附录C中描述的机器语言编写的程序存储在从地址30（十六进制）开始的主存储器中。当执行该程序时它会完成什么任务？

```
2003
2101
2200
2310
1400
3410
5221
5331
3239
333B
B248
B038
C000
```

29. 概述当附录C描述的机器执行一条操作码为B的指令时所涉及的步骤。用一组说明来表示你的答案，就像你在告诉CPU做什么。

*30. 概述当附录C描述的机器执行一条操作码为5的指令时所涉及的步骤。用一组说明来表示你的答案，就像你在告诉CPU做什么。

*31. 概述当附录C描述的机器执行一条操作码为6的指令时所涉及的步骤。用一组说明来表示你的答案，就像你在告诉CPU做什么。

*32. 假设在附录C描述的机器里，寄存器4和5中分别包括位模式3A和C8，在执行下列每条指令后，寄存器0中会留下什么位模式？

a. 5045　　b. 6045　　c. 7045
d. 8045　　e. 9045

*33. 利用附录C描述的机器语言，为完成下面每个任务分别编写一个程序。

a. 将存储单元44中存储的位模式复制到存储单元AA中。

b. 将存储单元34中的最低4个有效位变成0，并保持其他位不变。

c. 将存储单元A5中的最低4个有效位复制到存储单元A6中的最低4个有效位，并保持A6中的其他位不变。

d. 将存储单元A5中的最低4个有效位复制到存储单元A5中的最高4个有效位。（于是，A5中的前4位将和后4位相同。）

*34. 完成下列运算。

```
a.        111001     b.        000101
    AND   101001         AND   101010

c.        001110     d.        111011
    AND   010101         AND   110111

e.        111001     f.        010100
    OR    101001         OR    101010

g.        000100     h.        101010
    OR    010101         OR    110101

i.        111001     j.        000111
    XOR   101001         XOR   101010

k.        010000     l.        111111
    XOR   010101         XOR   110101
```

*35. 为了完成下面的任务，确定所需要的掩码和逻辑运算。

a. 将一个8位位模式的高4位置0，并且不影响其他位。
b. 将一个8位的位模式最高有效位取反，并且不影响其他位。
c. 将一个8位的位模式取反。
d. 将一个8位位模式的最低有效位置0。并且不影响其他位。
e. 将一个8位位模式的除最高有效位外所有的位都置1，并且不改变最高有效位。

*36. 确定一个逻辑运算（以及相应的掩码），使得当其用于一个8位的输入串时，当且仅当输入串为10000001时，产生的输出为全0位串。

*37. 描述一组逻辑运算（以及它们相应的掩码），使得当其用于一个8位的输入串时，当这个输入串的最高位和最低位都是1时输出结果为全0位串；否则输出中至少应包含一个1。

*38. 对下列位模式执行循环左移4位后，结果如何？

a. 10101　b. 11110000　c. 001
d. 101000　e. 00001

*39. 下列字节用十六进制记数法表示，对其执行循环右移2位后，结果如何？（用十六进制记数法写出答案。）

a. 3F　b. 0D　c. FF　d. 77

*40. a.在附录C描述的机器语言中，用什么样的单个指令可以完成寄存器B循环右移5位？
b.在附录C描述的机器语言中，用什么样的单个指令可以完成寄存器B循环左移2位？

*41. 用附录C描述的机器语言编写程序：把地址为8C的存储单元的内容颠倒过来。（也就是说，对于地址8C最后的位模式，从左向右读取与最初从右向左读取一致。）

*42. 用附录C描述的机器语言编写程序：将地址A2中存储的值减去A1中存储的值，并将结果存于地址A0中。假定值用二进制补码记数法编码。

*43. 高清视频可以以30 fps的速率传输，其中每一帧的分辨率为1920×1080像素，每像素使用24位。这种格式的无压缩视频流可以通过USB 1.1串行端口发送吗？USB 2.0串行端口呢？USB 3.0串行端口呢？（注意，USB 1.1、USB 2.0、USB 3.0串行端口的最大速度分别是12 Mbit/s、480 Mbit/s和5 Gbit/s。）

*44. 假设某人在键盘上每分钟能打40个单词。（假设一个单词以5个字符计。）如果计算机每微秒（百万分之一秒）执行500条指令，那么该计算机在此人打两个连续的字符之间可以执行多少条指令？

*45. 对于一个每分钟打40个单词的打字员，键盘每秒传输多少位才能跟得上？（假定每个字符以ASCII编码，每个单词以6个字符计。）

*46. 假设附录C中描述的机器与使用存储映射输入/输出技术的打印机通信，同时假设地址FF用于将字符发送给打印机，地址FE用于接收该打印机的状态信息。特别地，假设地址FE的最低有效位用于指示该打印机是否准备好接收下一个字符（0表示“未准备好”，1表示“准备好”）。从地址00开始，编写一个机器语言例程，它等待打印机准备好接收下一个字符，然后把由寄存器5中位模式表示的字符发送给打印机。

*47. 用附录C描述的机器语言编写一个程序：它在地址从A0到C0的所有存储单元中存放0，但是它应该足够小以致能够存放在地址从00到13（十六进制）的存储单元中。

*48. 假设某计算机硬盘上有200 GB存储空间可用，以15 Mbit/s的速率从宽带上接收数据。以这个速率，需要多久可以存满可用的存储空间？

*49. 假设某卫星系统正以250 Kbit/s的速率接收串行数据流。如果一个突发的大气干扰持续了6.96 s，那么有多少数据位会受到影响？

*50. 假设给你32个处理器，每个处理器在一秒钟内能够完成两个多位数字加法运算100万次。描述如何使用并行处理技术，使得能够在6×10^{-6} s的时间内完成64个数的求和。单独一个处理器完成相同的计算需要多少时间？

*51. 概述CISC体系结构和RISC体系结构之间的区别。

*52. 说出两种提高吞吐量的方法。

*53. 对于计算一组数值的平均值，说明在一台多处理器的计算机上为何比在一台单处理器的计算机上快得多？

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 假设某计算机生产商开发了一种新型的计算机体系结构。该公司在多大程度上可以拥有该体系结构所有权？什么样的政策对社会最好？
2. 从某种意义上说，1923年是现今被许多人称为有计划淘汰现象诞生的时间。这一年，由斯隆领导的通用汽车公司将汽车工业引向了型号概念的年代。其思想是通过改变风格，而不是必须推出更好的车来提高销售。引用斯隆的一句话："我们希望你们对自己现在的车不满意，于是你们将会购买新车。"如今，计算机行业在多大程度上使用了这种市场策略？
3. 我们常常在想，计算机技术如何改变了我们的社会。不过，许多人争辩说，这门技术常常抑制改变的产生，它试图使老系统继续存在，甚至使其地位更加牢固。例如，如果没有计算机技术，中央政府在社会中的角色会继续存在吗？如果没有计算机技术，中央集权在今天能够达到什么程度？如果没有计算机技术，我们在多大程度上会更好或更坏？
4. 如果某人认为自己不需要知道机器的任何内部细节（因为有其他人会建造它，维护它，并解决可能发生的问题），这种想法合理吗？你的答案会取决于这个机器是计算机、汽车、核电厂还是烤面包机吗？
5. 假设一家厂商生产了一种计算机芯片，但是后来发现它设计上有一个瑕疵。再假设该生产商在接下来的生产中修正了瑕疵，但决定掩盖最初有瑕疵这一事实，并且不回收已经售出的芯片，理由是：已经售出的芯片没有一个在该瑕疵会导致严重后果的应用中使用。有人会因为该生产商的决定受到伤害吗？如果没有人受到伤害，而且该决定避免了该生产商资金的流失亦或者避免了辞退员工，那么该生产商的决定正确吗？
6. 技术进步有助于治愈心脏病，还是会因导致久坐的生活习惯进而导致心脏病？
7. 很容易想象，由于溢出和截断错误而产生的算术差错可能会导致金融或导航方面的灾难。对于图像存储系统，由于丢失图像细节（也许在勘察或医疗诊断领域）而产生的错误会有什么后果？
8. ARM公司是一家为各种消费性电子设备设计处理器的小型公司。它并不制造任何处理器，而是将其设计授权给半导体厂商（如高通、三星和德州仪器），这些厂商为生产出的每个部件支付特许权使用费。这种商业模式将计算机处理器的高研发成本分散到了整个消费性电子市场。今天，95%以上的移动电话（不仅是智能手机）、40%以上的数码相机和25%的数字电视都使用ARM处理器。此外，ARM处理器还用于小型笔记本、MP3播放器、游戏控制器、电子书阅读器、导航系统等设备中。鉴于此，你是否认为该公司是垄断者呢？为什么是？为什么不是？随着消费类设备在今天的社会中扮演着越来越重要的角色，对于这一知名小型公司的依赖是好事吗？或者，是否会引起人们的担忧？

课外阅读

Carpinelli, J. D. *Computer Systems Organization and Architecture*. Boston, MA: Addison-Wesley, 2001.

Comer, D. E. *Essentials of Computer Architecture*. Upper Saddle River, NJ: Prentice-Hall, 2005.

Dandamudi, S P. *Guide to RISC Processors for Programmers and Engineers*. New York: Springer, 2005.

Furber, S. *ARM System-on-Chip Architecture*, 2nd ed. Boston, MA: Addison Wesley, 2000.

Hamacher, V. C., Z. G. Vranesic, and S. G. Zaky. *Computer Organization*, 5th ed. New York: McGraw-Hill, 2002.

Knuth, D. E. *The Art of Computer Programming*, vol. 1, 3rd ed. Boston, MA: Addison-Wesley, 1998.

Murdocca, M. J. and V. P. Heuring. *Computer Architecture and Organization: An Integrated Approach,* New York: Wiley,2007.

Stallings, W. *Computer Organization and Architecture*, 7th ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Tanenbaum, A. S. *Structured Computer Organization*, 5th ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

第3章

操作系统

这一章，我们将讨论操作系统。操作系统是用来协调计算机的内部活动以及检查计算机与外部世界通信的软件包。操作系统能将计算机硬件转化为有用的工具，我们的目标就是要理解操作系统做哪些工作，以及它是如何完成这些工作的。要成为有知识的计算机使用者，这样的背景是极为重要的。

本章内容

操作系统（operation system）是控制计算机整体运行的软件。它提供了用户可以存储和检索文件的方法，提供了用户可以请求执行程序的接口，还提供了执行被请求程序所必需的环境。

操作系统最著名的例子是Windows，微软公司已经发布了很多版本，并广泛用于PC机领域。另一个被广泛认可的例子是UNIX，它是服务于较大的计算机系统和PC群的流行选择。事实上，UNIX是其他两个操作系统的核心：Mac OS是苹果公司为其一系列Mac机提供的一种操作系统，Solaris由Sun Microsystems（现归Oracle所有）开发得来。另外，还有能够运用于大型机和小型机的Linux操作系统，该系统最初是由一些计算机爱好者以非盈利的目的开发的，到目前为止，包括IBM公司在内的许多商业机构都发布了Linux操作系统。

对于非专业的计算机用户，他们只能感觉到不同操作系统表面上的不同，而对于计算机专业人员来说，不同的操作系统可能意味着使用完全不同的工具或在传播和维护工作中遵循完全不同的理念。然而，所有主流操作系统的核心都是解决计算机专家在半个多世纪之前就已经遇到的那些问题。

3.1 操作系统的历史

今天的操作系统经过长期的演变已经成为大而复杂的软件包。20世纪四五十年代，计算机不是很灵活，效率也不高。一台计算机会占据整个房间。执行程序需要大量的设备准备工作，如安装磁带、把穿孔卡片放在读卡机上、设置开关等。每个程序的执行称为一个**作业**（job），它是作为一个独立的活动处理的——为执行该程序准备好计算机，执行程序，然后在下一个程序的准备工作开始之前，必须重新获取磁带、穿孔卡片等所有一切。当几个用户需要共享一台机器时，操作系统提供签名表，以便各个用户能够预订到一段机器时间。在分配给某个用户的时间段内，机器就完全处于该用户的控制之下。这段时间通常是从程序的准备开始，接下来是

短时间的程序执行过程。一个用户本可以在很短的时间内尽可能多做一件事情（“它仅需一分钟”），但下一个用户已经迫不及待地要使用机器做准备工作了。

在这样的环境下，操作系统开始作为一个系统致力于简化程序的准备工作，提高作业之间的过渡效率。操作系统早期的开发是用户与设备的分离，用以避免用户进出计算机机房。为此雇用了计算机操作员来操作机器。任何人如果需要运行程序，就必须把程序、所需的数据以及有关程序需求的特别说明提交给操作员，由操作员返回结果。操作员所做的工作就是把这些资料输入到计算机的海量存储器，然后由称为操作系统的程序从那里一次一个地读入并执行程序。这就是**批处理**（batch processing）的开始——把若干个要执行的作业收集到一个批次中，然后执行而无需与用户发生进一步的交互。

在批处理系统中，驻留在海量存储器中的作业在**作业队列**（job queue）里等待执行（见图3-1）。**队列**（queue）是一种存储机构，对象（这里指作业）按照FIFO（first-in，first-out，先进先出）的方式在队列里排队。也就是说，对象的出列顺序和入列顺序一致。实际上，大多数作业队列不是严格遵循FIFO结构的，主要是因为大多数操作系统都考虑了作业的优先级，结果就造成了在队列中等待的作业有可能被优先级更高的作业挤掉。

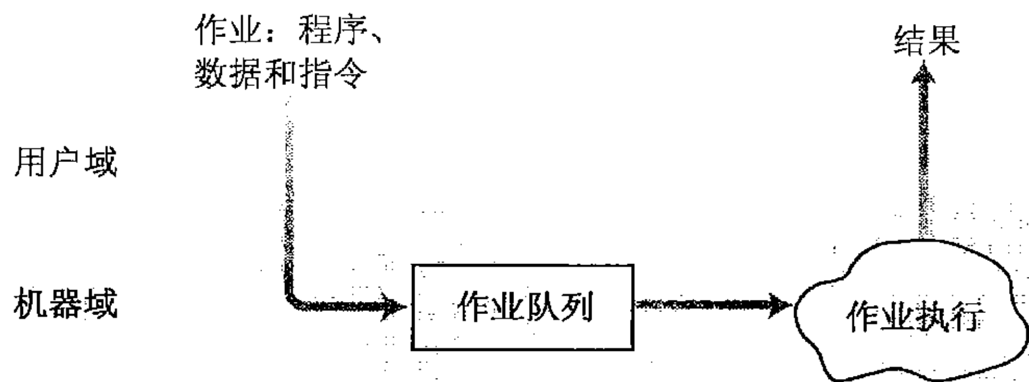


图3-1 批处理

在早期的批处理系统中，每个作业都伴随着一组指令，用来说明为这个特定的作业准备机器时所需的步骤。这些指令用作业控制语言（JCL）进行编码，与作业一起存放在作业队列里。当一个作业被选中执行时，操作系统在打印机上打印出这些指令以便计算机操作员阅读和遵照执行。在今天，计算机操作员与操作系统之间的通信仍然存在，如报告“磁盘驱动不可访问”和“打印机没有响应”之类的错误的PC操作系统。

在计算机和用户之间，用计算机操作员作为媒介的最大缺点是：作业一旦提交给操作员，用户就与它无法交互。这种方法对于某些应用是可以接受的，如工资表的处理，因为在这里，数据与所有的处理决策事先已经建立了。然而，当在一个程序的执行期间，用户必须与该程序进行交互时，这种方法就无法让人接受了。例如，在预订系统中，预订和取消操作必须及时报告；在字处理系统中，文档是以动态的写入和重写方式开发的；在计算机游戏中，与计算机的交互性是游戏的主要特征。

为了适应这些需求，人们开发了新的操作系统，它们允许执行一个程序来通过远程终端与用户对话——这种特性称为**交互式处理**（interactive processing）（见图3-2）。一个终端只不过是一台电子打字机，通过电子打字机用户能够进行输入并且读出那些打印在纸上的计算机响应。当今的终端已经演变成称为工作站的设备（这些设备更为精细复杂），而且在必要时甚至还可以是一台完全独立运行的完整个人电脑。

成功交互式处理的最重要之处在于，计算机的动作能够足够快速地协调用户的需求，而不是让用户遵循计算机的时间表。（在进行工资表的处理时，计算机能够根据所需的时间量调度得很好，但是在使用字处理程序时，如果机器不能敏捷地对字符的打印作出响应，用户会很沮丧。）

从某种意义上说，计算机在一个限期内被强制执行任务，这一过程就是众所周知的**实时处理**（real-time processing），并且动作的完成也是按实时方式发生的。也就是说，要是说计算机以实时的方式完成一个任务，就意味着计算机完成任务的速度足以跟上该任务所在的外部（现实世界）环境中的行为。

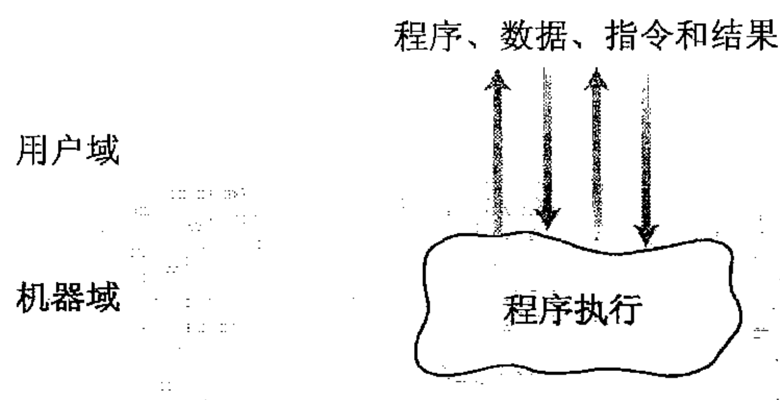


图3-2 交互式处理

如果要求交互式系统一次只服务于一个用户，那么实时处理就不存在问题了。但是在20世纪六七十年代，计算机比较昂贵，因此每台计算机不得不服务于多个用户。因此，工作在终端的若干个用户在同一时间寻求一台机器的交互式服务，并且对实时的要求出现障碍就不足为奇了。如果操作系统对于多用户环境仍然坚持一次执行一个作业，那么将只有一个用户可接受到满意的实时服务。

针对这个问题的解决方案就是设计能同时给多个用户提供服务的操作系统，这一特点称为**分时**（time-sharing）。实现分时的一种方法就是应用称为**多道程序设计**（multiprogramming）的技术，其中时间被分割成时间片，每个作业的执行被限制为每次仅一个时间片。在每个时间片结束时，当前的作业暂时放弃执行，允许另一个作业在下一个时间片里执行。通过这种方法可以快速地在各个作业之间进行切换，形成了若干个作业同时执行的假象。依据所执行的作业的类型，早期的分时系统能够同时为多达30个用户提供可接受的实时服务。今天，多道程序设计既可用于单用户系统，也可以用于多用户系统，前者通常称为**多任务**（multitasking）。也就是说，分时指的是多个用户共享对同一计算机的访问，而多任务指的是一个用户同时执行多个任务。

随着多用户的发展，分时操作系统作为一种典型配置，被用在大型的中央计算机上，用来连接大量的工作站。通过这些工作站，用户能够从机房外面直接与计算机进行通信，而不用把请求递交给计算机操作员。通常把常用的程序存储在计算机的海量存储设备上，然后通过操作系统来响应工作站的请求并执行这些程序。这样，作为计算机与用户中间媒介的计算机操作员作用就不那么明显了。

到今天，特别是在个人计算机领域，计算机用户已经能够承担计算机操作的所有职责。因此，计算机操作员在事实上已经不存在了，即使大型计算机系统，其运行也基本上无须人工参与。事实上，传统的计算机操作员已经让位于系统管理员，系统管理员管理计算机系统，获得和监控计算机新设备和软件的安装，实施一些本地的规则，例如建立新的账号、为不同的用户划分一定的存储容量、协调用户一起解决系统中出现的问题，这样就比纯手工方式操作计算机要好得多。

总之，操作系统已经从简单的一次获取和执行一个程序发展为能够分时处理，能够管理计算机海量存储设备上的程序和数据文件，并能直接响应计算机用户请求的复杂系统。

但是，计算机操作系统的发展仍在继续。多处理器机器的发展已经让操作系统能够进行分时／多任务处理，操作系统把不同的任务分配给不同的处理器进行处理，以及采用分时机制共享单个处理器。这些操作系统必须处理**负载平衡**（load balancing，动态地把任务分配给各个处理器，使得所有处理器都得到有效的利用）和**均分**（scaling，把大的任务划分为若干个子任务，

并与可用的处理器数目相适应）问题。

此外，计算机网络的出现（相距很远的大量计算机连接在一起）使得有必要发展相应的软件系统来规范网络的行为。计算机网络领域（我们将在第4章学习这部分内容）在许多方面拓展了操作系统这个学科，其目标是跨多个用户和计算机（而非单一的、孤立的计算机）管理资源。

操作系统的另一个研究方向侧重专用于特定任务的设备，如医疗设备、车载电子设备、家用电器、手机或其他手持电脑。这些设备中的计算机系统称为**嵌入式系统**（embedded system）。嵌入式操作系统通常能够节省电池电量、满足严格的实时截止时间，或在只有很少或完全没有人为监管的情况下连续工作。在这些努力中，有代表性的成功系统有：VxWORKS，它由Wind River系统公司开发，在称为“精神和机会”的火星探索旅程中也发挥了作用；Windows CE（也就是众所周知的Pocket PC），它由微软开发；Palm OS，它由PalmSource公司开发，主要用在手持设备上。

智能手机有什么功能

手机的功能越来越强大，除简单处理语音通话之外还能提供其他服务。现在，常见的**智能手机**（smartphone）可以用于编写文本消息、浏览万维网、导航、查看多媒体内容。简而言之，它可以提供许多传统的个人计算机所能提供的服务。因此，智能手机需要成熟的操作系统，不仅用来管理有限的智能手机硬件资源，还要支持迅速增加的智能手机应用软件。智能手机操作系统市场的争夺之战将日趋激烈，而争夺的焦点很有可能在于哪个操作系统可以以最优的价格提供最有创意的功能。智能手机操作系统领域中的竞争在苹果公司的iPhone操作系统、黑莓手机生产商Research In Motion的BlackBerry操作系统、微软的Windows Phone、诺基亚的Symbian操作系统和谷歌的Android之间展开。

问题与练习

1. 举出几个队列的例子。对于每一种情况，请指出任何可能破坏FIFO结构的情况。
2. 下列任务中哪些需要用到实时处理技术？
   a. 打印邮件列表。
   b. 玩计算机游戏。
   c. 拨号时，把这些数字显示在智能手机屏幕上。
   d. 执行一个预报下一年经济状况的程序。
   e. 播放MP3录音。
3. 嵌入式系统与PC的区别是什么？
4. 分时处理与多任务处理的区别是什么？

3.2 操作系统的体系结构

为了能够理解一个典型操作系统的组成，这里我们首先考虑一个典型的计算机系统中有哪些软件，软件是如何分类的，然后我们再回到操作系统上来。

3.2.1 软件概述

我们通过提出一个软件分类方案考察一个典型计算机系统中的软件。这种分类方案总是把一些类似的软件单元放在不同的类里，其方法如同时区的划分。（时区的划分使得相邻时区的设

置相差一小时，即使其日出与日落的时间没有明显的差别。）其次，在软件分类的情况下，学科的发展变化和某种权威的缺乏，导致了一些矛盾的分类方法。例如，微软公司Windows操作系统的用户会发现，“附件”和“管理工具”程序组，既包括应用类软件又包括实用类软件。因此，下面的分类方法应该被看做在广泛的、动态的学科里占有一席之地的工具，而不是对人们普遍接受的事实的一种表述。

先把计算机软件分为两大类：**应用软件**（application software）和**系统软件**（system software）（参见图3-3）。应用软件是由一些完成计算机特定任务的程序组成的。一台用来维护某个制造公司库存单的计算机所包含的应用软件与电气工程师用的计算机里的软件是不同的。应用软件的例子有电子制表软件、数据库系统、桌面出版系统、记账系统、程序开发软件以及游戏等。

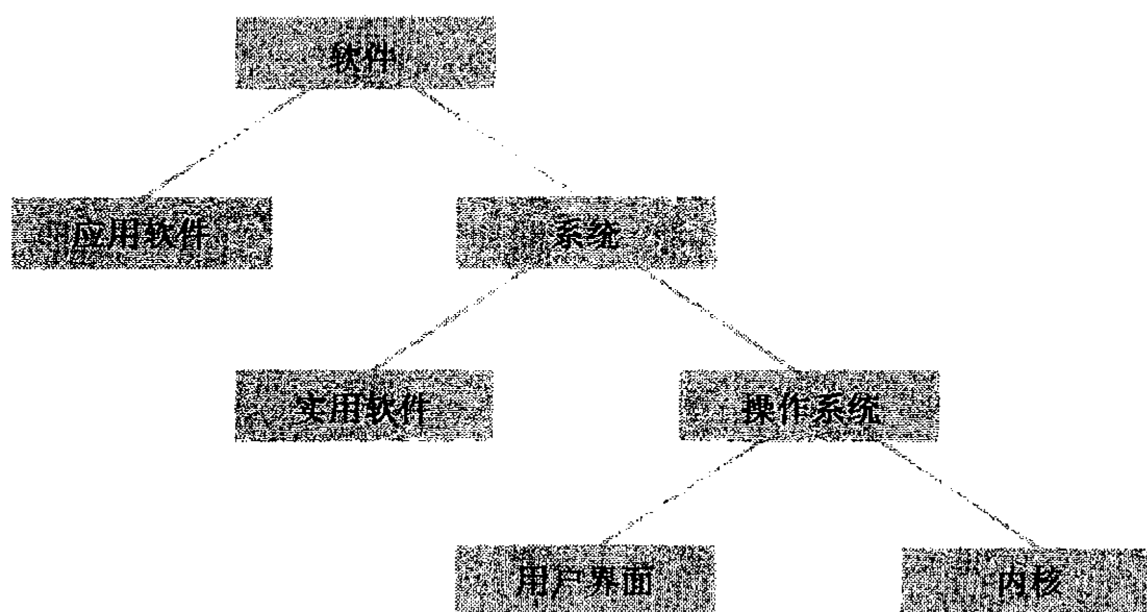


图3-3 软件分类

相对于应用软件而言，系统软件完成一般的计算机系统都需要完成的任务。在某种意义上，系统软件提供了应用软件所需要的基础架构，这和国家基础架构（政府、道路、公共设施、金融机构等）提供公民维系各自生活方式的基础的方式大致相同。

系统软件又可分两类，一类是操作系统本身，另一类是统称为**实用软件**（utility software）的软件单元。安装的大多数实用软件包括这样一些程序，它们实现的活动仅仅是计算机的安装的基础，而没有包含在操作系统中。从某种意义上说，实用软件是由一些能够扩充（或定制）操作系统功能的软件单元组成的。举例来说，格式化磁盘或将文件从磁盘复制到光盘中去的能力仅仅是借助于实用软件，而不是在操作系统内部实现的。其他的实用软件的例子包括数据压缩与解压缩软件、多媒体播放软件和处理网络通信的软件。

把某些工作作为实用软件来实现，允许定制系统软件，这比把它们交给操作系统来执行要更容易满足特定安装的需求。事实上，一些公司和个人对原先和计算机操作系统一起提供的实用软件进行修改和扩充，已经是很普通的事情了。

遗憾的是，应用软件与实用软件之间的差别已经很模糊。从我们的观点来看，它们的差别在于其是否是计算机软件架构的一部分。因此，当新的应用变成了一种基础的工具，这个应用就很可能成为一种实用软件。当用于因特网的通信软件还在研究阶段时，它就被认为是一种应用软件，而在今天，像这样的工具软件对大部分PC机应用而言非常基础，也就被定义为了实用软件。

实用软件和操作系统的差别同样是模糊的。特别是，美国和欧洲的反垄断诉讼案争论的都是这样一个问题：浏览器和媒体播放器这两个组件是微软公司操作系统的一部分，还是微软公司用来压制竞争对手的实用软件。

Linux

对于计算机爱好者而言，如果想通过亲手实验来了解一个操作系统，那么就应该选择Linux。最初的Linux操作系统是由Linus Torvalds在赫尔辛基大学学习期间设计的。Linux操作系统是一个非专利产品，我们可以免费获得它的源代码（见第6章）和相关文档。因为可以免费获得其源代码，所以该系统在计算机爱好者、学习操作系统的学生和程序员中非常流行。而且，Linux操作系统被认为是当今可用的较可靠的操作系统之一。正因为这个原因，一些公司开始以更实用的形式包装和销售Linux操作系统产品，现在这些产品开始向市场上长期被认可的商用操作系统产品发出了挑战。我们可以在http://www.linux.org这个网站了解更多有关Linux的知识。

3.2.2 操作系统组件

现在，我们把注意力集中在操作系统领域内的组件上。为了完成计算机用户请求的动作，操作系统必须能够与这些用户进行通信。操作系统负责处理这种通信的部分通常称为**用户界面**（user interface）。老式的用户界面称为shell，通过键盘和显示屏用文本信息与用户通信。更现代化的系统利用GUI（Graphical User Interface，图形用户界面）实现与用户的通信，其中操作的对象（如文件和程序）被表示为显示屏上的图标（icon）。这些系统允许用户使用某种常用的输入设备发出命令。例如，有一个或多个按钮的计算机鼠标可用来单击或拖曳屏幕上的图标。另外，平面设计师或某些类型的手持设备常使用专用的指示设备或手写笔代替鼠标来操作图标。最近，高密度触摸屏的进步使得用户可以直接用手指操作图标。当今的GUI使用二维图像投影系统，三维立体界面允许用户通过3D投影系统、知觉设备和环绕音频再生系统与计算机进行通信，它是当前研究的课题。

虽然操作系统的用户界面在实现计算机的功能上扮演了重要的角色，但它仅仅是用户与操作系统内核之间的一个接口而已（见图3-4）。用户界面与操作系统内部之间的区别的呈现是因为这样一个事实，即一些操作系统允许特定用户从各种界面中选择最合适的界面为自己服务。例如，UNIX操作系统的用户就可以选择不同的shell，包括Bourne shell、C shell和Korn shell，以及称为X11的GUI。最早的Microsoft Windows是一个GUI应用程序，可以通过MS-DOS操作系统的shell命令加载。在最新版Windows中，人们仍可看见作为实用程序存在的DOS shell cmd.exe，但非专业用户几乎完全不需要使用这一界面。类似地，苹果公司的OS X保留了一个Terminal实用软件（utility shell），它承袭了系统的UNIX shell。

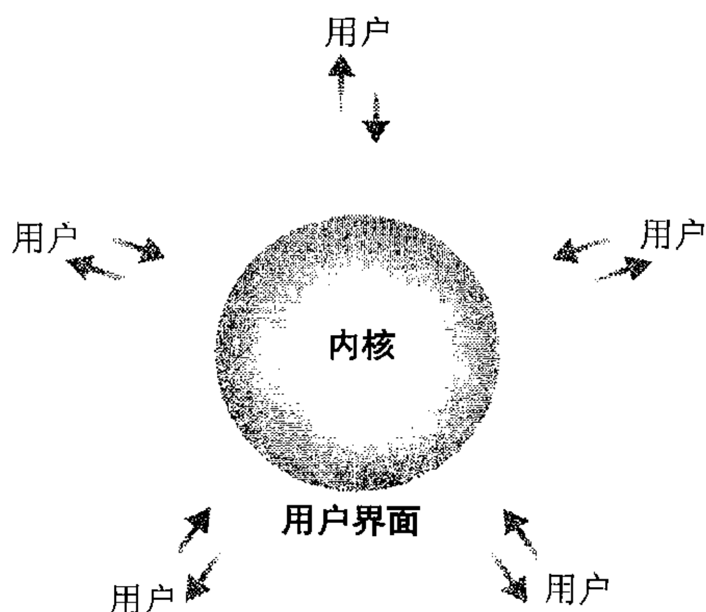


图3-4 作为用户和操作系统内核之间中介的用户界面

今天的GUI shell中的重要组件是**窗口管理程序**（window manager），该程序在屏幕上分配若干块（称为窗口的区域），并且跟踪与每个窗口相联系的应用程序。当一个应用程序想在屏幕上显示图像时，它就会通知窗口管理程序，窗口管理程序就会把图像放在分配给该应用程序的窗口里。然后，当单击鼠标时，窗口管理程序计算鼠标在屏幕上的位置，并把这个鼠标动作通知给相应的应用程序。窗口管理程序负责生成GUI样式，且大多数管理程序会提供一系列配置选项。Linux用户甚至可以选择窗口管理程序，常用的选项包括KDE和Gnome。

与操作系统的用户界面相对，我们把操作系统内部的部分称为**内核**（kernel）。操作系统的内核包含一些完成计算机安装所要求的极基本功能的软件组件。其中一个组件是**文件管理程序**（file manager），它的工作是协调计算机海量存储器设施的使用。更准确地说，文件管理程序维护着存储在海量存储器上的所有文件的记录，包括每个文件的位置、哪些用户有权访问各种文件以及海量存储器里的哪部分可以用来建立新文件或扩充现有文件。这些记录被存放在单独的包含相关文件的存储介质中，这样，每次存储介质启动时，文件管理程序就能够检索相关的文件，进而就能知道特定的存储介质中存放的是什么。

为了方便计算机用户，大多数文件管理程序都允许把若干个文件组织在一起，放在**目录**（directory）或**文件夹**（folder）里。这种方法允许用户将自己的文件依据用途划分，把相关的文件放在同一个目录里。一个目录可以包含称为子目录的其他目录，这样就可以构建层次化的目录结构。例如，用户可以创建一个名为MyRecords的目录，它又包含了名为FinancialRecords、MedicalRecords和HouseHoldRecords的3个子目录。每个子目录中都会有属于该范畴的文件。（Windows操作系统的用户能通过执行Windows资源管理器程序，让文件管理程序显示当前所有的目录结构。）

一条由目录内的目录所组成的链称为**目录路径**（directory path）。路径通常是这样表示的：列出沿该路径的目录，然后用斜杠分隔它们。例如，路径animals/prehistoric/dinosaurs表示的是：该路径是从目录名为animals的目录开始的，经过名为prehistoric的子目录，终止于名为dinosaurs的子目录。（对于Windows用户而言，目录路径是用反斜杠表示的，如animals\prehistoric\dinosaurs。）

其他软件实体对文件的任何访问都是由文件管理程序来实现的。该访问过程是这样开始的，先通过一个称为打开文件的过程来请求文件管理程序授权访问该文件，如果文件管理程序批准了该访问请求，那么它就会提供查找和操控该文件所需的信息。

内核的另外一个组件是一组**设备驱动程序**（device driver）。它们是负责与控制器（有时直接与外围设备）通信，以操作连接到计算机的外围设备的软件组件。每个设备驱动程序都是专门为特定类型的设备（如打印机、磁盘驱动器和显示器等）设计的，它把一般的请求翻译为这种设备（分配给这个驱动程序的设备）所需要的更富技术性的步骤。例如，打印机的设备驱动程序包含的软件能够读取和解码特定打印机的状态字，而且还能够处理其他一些信息交换的细节。这样，其他软件组件就没有必要为了打印一个文件处理这些技术细节，而只需要运用设备驱动程序软件完成打印文件的任务，技术细节交由设备驱动程序处理。按照这种方式，其他软件组件的设计可以独立于具体设备特有的特征。这样做的结果是，一个普通的操作系统能够使用一些特殊外围设备，我们只要安装合适的设备驱动程序即可。

在操作系统的内核中，还有一个组件就是**内存管理程序**（memory manager），它担负着协调和管理计算机使用主存储器的任务。在计算机一次仅执行一个任务的环境中，这些工作就比较简单了。这些情况下，执行当前任务的程序放在主存储器中已经定义好的位置上执行，然后被执行下一个任务的程序替换。然而，在多用户和多任务的环境下，要求计算机在同一时刻能够处理多个需求，这时内存管理程序的职责就扩展了。在这些情况下，许多程序和数据块必须同时驻留在内存里。因此，内存管理程序必须找到并给这些需求分配内存空间，并且要保证每个

程序只能限制在程序所分配的内存空间内运行。此外，随着不同活动的需求进出内存，内存管理程序必须能跟踪那些不再被占用的内存区域。

当所需的总内存空间超过该计算机实际所能提供的可用内存空间时，内存管理程序的任务会更复杂。在这种情况下，内存管理程序在内存与海量存储器之间来回切换程序和数据［称为**页面调度**（paging）］，这样就造成了有额外内存空间的假象。例如，假设需要一块8 GB的内存空间，但是计算机所能提供的只有4 GB。为了造成具有更大内存的假象，内存管理程序在磁盘上预留了4 GB的存储空间。在这块存储区域里，将记录内存实际容量有8 GB时本应存储在内存中的位模式。这块数据区被分成大小一致的存储单元，该存储单元称为**页面**（page），典型的页面大小只有几千字节。于是，内存管理程序就在主存和海量存储器之间来回切换这些页面。这样，在任何给定的时间内，我们所需的页面都会出现在4 GB的内存之中，最后的结果是计算机能够像确实拥有8 GB内存一样工作。这块由分页技术所产生的大的“虚构的”内存空间称作**虚拟内存**（virtual memory）。

另外，在操作系统内核中还有**调度程序**（scheduler）和**分派程序**（dispatcher）这两个组件，我们将在下一节介绍。在此，我们只需注意，在多道程序设计系统中调度程序决定哪些活动是可以执行的，而分派程序控制给这些活动的时间分配。

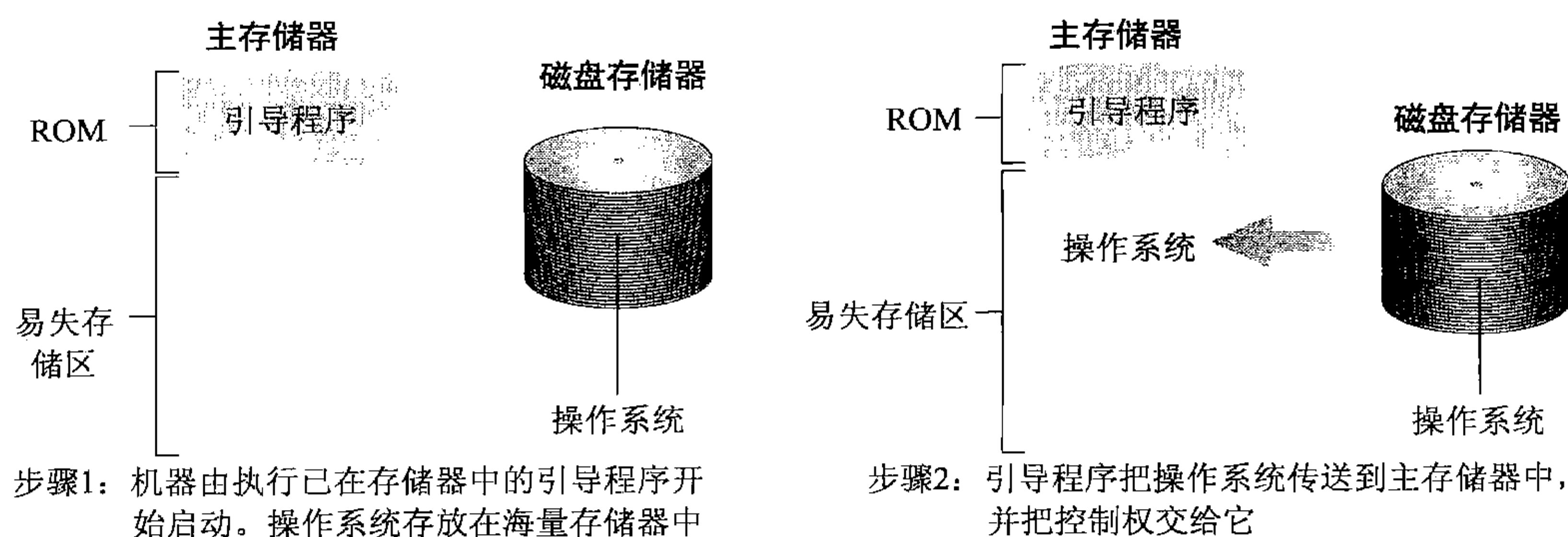
3.2.3　系统启动

我们已经可以看出，操作系统提供了其他软件组件所需的软件基础设施，但是我们还没有细想操作系统本身是如何启动的。这是通过一个称为**引导**（boot strapping，简称为booting）的过程实现的，这个过程是由计算机在每次启动的时候完成的。正是这个过程把操作系统从海量存储器（它永久存放的地方）传送到主存储器（在开机时，内存实际上是空的）中。为了理解启动过程和必须有启动过程的原因，我们先来考察计算机的CPU。

CPU的设计使得每次启动时，程序计数器都从事先确定的特定地址开始。CPU就在这个地址上期望能找到程序要执行的第一条指令。从概念上讲，只需在这个地址上存储操作系统。然而，从技术上讲，计算机的主存通常是采用易失性技术制造的，当计算机关闭时，也就意味着存储在内存上的数据会丢失。因此，在每次重启计算机的时候，我们就需要一种重新填充主存的方法。

简言之，当计算机首次打开时，我们需要在主存储器中有一个程序（最好是操作系统），但是每次关机后，计算机的易失性存储器都会被擦除。为了解决这个两难问题，计算机的一小部分主存储器就用非易失性存储单元建造，而这地方正是CPU期望找到初始程序的地方。由于这种存储器的内容可以读取，但不可以改变，因而被称为**只读存储器**（ROM）。打个比方，虽然所使用的技术是更先进的，但我们可以把在ROM中存储位模式想象成熔断微小的保险丝（熔断的表示1，未熔断的表示0）。更确切地说，如今个人电脑中大多数的ROM是用闪存技术构建的（即不是严格意义上的ROM，因为它可以在特定情况下被改变）。

在一般的电脑中，**引导程序**（boot loader）被永久存储在机器的ROM中。这样，在计算机开机的时候将最先执行这个程序。引导程序的任务是引导CPU把操作系统从海量存储器中预先定义的位置调入主存的易失性存储区（如图3-5）。现代的引导装载程序可以复制各种位置的操作系统到主存储器。例如，在嵌入式系统如智能手机中，操作系统是从闪存（非易失性存储器）复制的；在大型公司或大学的小型工作站上，可能要通过网络从远程机器上复制操作系统。一旦操作系统被放入主存，引导程序就引导CPU执行跳转指令，转到这个存储区。这时，操作系统接管并开始控制计算机的活动。执行引导程序和启动操作系统的整个过程称作**启动**（booting）计算机。



步骤1：机器由执行已在存储器中的引导程序开始启动。操作系统存放在海量存储器中

步骤2：引导程序把操作系统传送到主存储器中，并把控制权交给它

图3-5 引导过程

固 件

除了引导程序外，PC机的只读存储器还包括了一组例行程序，用于实现基本的输入/输出活动，如从键盘上接收信息、把信息显示在计算机的屏幕上，以及从海量存储器上读数据等。因为存放在非易失性存储器（如FlashROM）中，所以这些例行程序并不会不变地固化到机器的硅片（硬件）中，也不像海量存储中的其他程序（软件）那样随时可被更改。人们创造了**固件**（firmware）这个词来描述这一“中间地带”。固件例行程序可以被引导程序使用，以便在操作系统开始工作前完成I/O活动。例如，它们会在引导过程真正开始前，用于与计算机用户通信，并在引导期间提交错误报告。得到广泛使用的固件系统包括PC中一直使用的BIOS（Basic Input/Output System，基本输入/输出系统）、较新的EFI（Extensible Firmware Interface，可扩展固件接口）、Sun公司的Open Firmware（现为Oracle的一个产品），以及用于许多嵌入式设备的CFE（Common Firmware Environment，通用固件环境）。

你也许在想，为什么台式计算机不提供足够的ROM来装载整个操作系统呢，这样从海量存储器来引导启动就没有必要了。虽然对于使用小型操作系统的嵌入式系统而言这是可行的，但就当今的技术而言，把通用计算机的大块主存专用于非易失性的存储，效率就不高了。另一方面，计算机操作系统要频繁地进行更新，以确保安全并与最新硬件改良了的新设备驱动程序同步。虽然也有可能去更新存储在ROM中的操作系统和引导装载程序——通常称为**固件更新**（firmware update），但技术上的限制使得海量存储器成为了较传统的计算机系统的最普遍选择。

最后，我们要指出，理解引导过程以及操作系统、实用软件和应用软件之间的区别，能帮助我们更好地领会大多数通用计算机系统操作的运行方法。当这样的计算机第一次开机时，引导过程装入并激活操作系统，然后用户向操作系统提出请求，执行实用软件或应用程序。当实用软件或应用程序终止时，用户再次获得与操作系统的联系，这时用户能提出另一次请求。因此，学习使用这样的系统是一个双层过程，除了学习指定实用软件或期望的应用程序的细节之外，还必须学习足够多的关于计算机操作系统的知识，以便能游刃有余地切换应用程序。

问题与练习

1. 列举典型操作系统的组件，并用一句话概括每个组件的作用。
2. 应用软件与实用软件之间的区别是什么？
3. 什么是虚拟存储器？
4. 概述引导过程。

3.3 协调机器的活动

本节我们讨论操作系统如何协调应用软件、实用软件以及操作系统自身内部单元的执行。首先，从进程的概念开始。

3.3.1 进程的概念

现代操作系统的一个最基本概念就是程序与执行程序的活动之间的区别。前者是一组静态的指令，而后者是一个动态的活动，其属性会随着时间的推进而改变。（我们可以把程序想象成静静夹在架子上的一页乐谱，而把活动想象成演奏这段音乐的音乐家。）在操作系统的控制下执行某个程序的活动称为**进程**（process）。与进程联系在一起的活动的当前状态称为**进程状态**（process state）。这个状态包含正在执行的程序的当前位置（程序计数器的值）、CPU中其他寄存器的值以及相关的存储单元。大约说来，进程状态就是机器在特定时刻的快照。在程序执行期间的不同时刻（一个进程中的不同时刻），将观察到不同的快照（不同的进程状态）。

与音乐家一次仅尝试演奏一份音乐作品不同，在典型的分时/多任务计算机中通常会有许多进程同时在执行并竞争计算机资源。操作系统的任务就是管理这些进程，使每个进程都能获得其需要的计算机资源（外围设备、主存空间、访问文件以及访问CPU），确保独立进程不会相互干扰，确保需要交换信息的进程能够进行信息交换。

3.3.2 进程管理

与协调进程的执行有关的任务是由操作系统内核中的调度程序和分派程序处理的。调度程序维护一个有关计算机系统中现存进程的记录（也就是进程池），将新的进程加入到该进程池中，并把已经完成的进程移出进程池。这样，当用户请求执行一个应用时，调度程序就把这个应用加到当前进程池加以执行。

为了跟踪所有的进程，调度程序在主存中维护着一个信息块，称为**进程表**（process table）。每当要请求程序执行时，调度程序都在进程表中为该程序创建一个新的表项。这个表项包含有如分配给该进程的存储区（从内存管理程序得到）、进程的优先级以及该进程是处于就绪状态还是等待状态这样的信息。如果进程能够继续执行，那么该进程就处于**就绪**（ready）状态；如果进程因为要等待某个外部事件的发生而中断，例如海量存储操作的完成、等待键盘的按键以及等待其他进程传来的消息等，那么该进程就处于**等待**（waiting）状态。

分派程序是内核的一个组件，它确保被调度的进程能实际执行。在分时/多任务系统中，这个任务是依靠**多道程序设计**（multiprogramming）来完成的，也就是说，先将时间划分为小的时间段，每段称为一个**时间片**（time slice），通常毫秒或微秒计量，然后把CPU的注意力放在就绪进程上，允许每个进程一次执行一个时间片（参见图3-6）。这种从一个进程到另一个进程的改变过程称为**进程切换**（process switch）或进程**上下文切换**（context switch）。

每次分派程序给进程分配一个时间片，它都会初始化一个计时器电路，通过产生一个**中断**（interrupt）信号来指示时间片的结束。CPU对中断信号的响应方法就如同你被一个任务打断时的应对方法，你停止当时正在做的工作，记录当时任务进展的位置（这样就能在以后返回到被中断工作的接续位置），然后处理中断事件。当CPU收到一个中断信号时，它会完成当前的机器周期，保存它在当前进程中的位置，然后就开始执行称为**中断处理程序**（interrupt handler）的程序，该程序存放在主存中的预先定义的位置上。中断处理程序是分派程序的一部分，它用来描述分派程序如何响应中断信号。

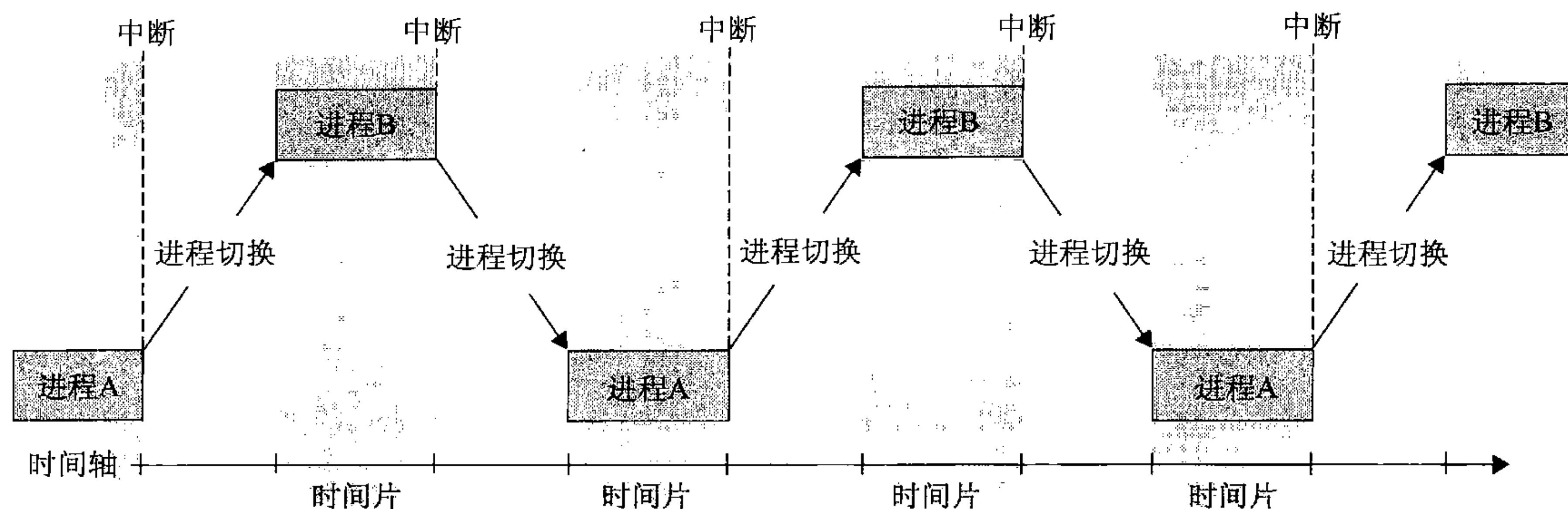


图3-6 进程A与进程B之间的多道程序设计

中 断

中断是用来终止时间片的，正如本文中所描述的一样，这只是计算机中断系统中众多应用中的一个。有许多可以产生中断信号的环境，每个都有自己的中断例程。事实上，中断为协调计算机的活动与相关环境提供了一个重要的工具。例如，单击鼠标和按下键盘中的一个按键都能产生一个中断信号，这样就能使CPU放下正在处理的工作，转而去解决中断。

为了管理识别和响应中断的任务，不同的中断信号被赋予了不同的优先级，这样一来，最重要的任务最先得到关注。最高级别的中断通常与电源故障有关，像计算机电源意外中断而产生的中断信号等。然后，在几毫秒时间内，与之相关的中断处理例程赶在电压降到不能再进行操作前，引导CPU完成一系列类似“家务活”的繁杂工作。

于是，中断信号的作用就是取代当前进程，将控制权传回分派程序。此时，分派程序从进程表的就绪队列（由调度程序决定）中选择优先级最高的进程，重启计时器电路，使被选择的进程开始它的时间片加以执行。

多道程序设计系统能够成功的最大关键是能够停止进程，并且稍后能重启进程。如果你在读一本书的时候被打断了，那么你能否在稍后继续阅读就依赖于你是否记得中断时读到的位置，以及你是否记得那个位置之前的内容。简而言之，你必须能够重新建立起中断前所存在的那个环境。

在一个进程的情况下，必须重新建立的环境就是进程的状态。回想一下，这个状态包括程序计数器的值以及寄存器和相关存储单元的值。在为多道程序设计系统开发的CPU中，保存这种信息的任务是CPU应对中断信号的工作的一部分。这类CPU还提供机器语言指令，以重新装入先前保存的状态。这种特性简化了分派程序完成进程切换的任务，它也例证了现代的CPU设计是如何受当今操作系统的需求影响的。

最后，我们应当注意到，多道程序设计是为了提高计算机总体效率的。这有点违反常理，因为多道程序设计要对进程来回切换，因而会产生一定的开销。但是，如果没有多道程序设计处理技术，那么每个进程在下一个进程开始之前完成执行，这也就意味着进程等待外围设备来完成任务，或者用户发出下一个请求的时间被浪费了。多道程序设计技术可以把这些丢失的时间给其他进程。例如，如果一个进程执行I/O请求，如向磁盘提出读数据请求，那么调度程序就会更新进程表来反映出这个进程正在等待外部事件。结果是，分派程序将不再给该进程分配时间片。之后（也许是几百毫秒），当I/O请求完成时，调度程序将会更新进程表来显示该进程处于就绪状态，这样这个进程就可以重新竞争时间片了。简而言之，当正在执行I/O请求时，程序

可以执行其他的任务，因此一组任务的完成时间要比按照顺序方式执行所花的时间少。

问题与练习

1. 概述程序和进程的差别。
2. 概述在中断出现时，CPU要完成哪些步骤。
3. 在多道程序设计系统中，如何能使高优先级的进程运行得比其他进程快？
4. 在一个多道程序设计系统里，如果每个时间片是50 ms，每次上下文切换所花费的时间最多是1 μs，那么计算机在1 s内能够服务于多少个进程？
5. 在练习4中，如果每个进程都完全使用了它的时间片，那么实际花费在进程执行上的时间占整个机器时间的比例是多少？如果每个进程在它的时间片后的1 μs执行I/O请求，那么这个比例又是多少？

*3.4　处理进程间的竞争

操作系统的一个重要任务就是将机器的各种资源分配给系统中的各个进程。从广义上讲，我们所用的**资源**（resource）这个术语，不仅包括机器的外围设备，还包括机器本身的特性。文件管理程序分配对文件的访问并为新建立的文件分配磁盘空间，内存管理程序分配内存空间，调度程序分配进程表的空间，分派程序分配时间片。正如计算机系统里的许多问题一样，这种分配任务表面上看起来很简单，但实际上，对于没有设计好的操作系统，几个微小的错误将导致系统的故障。要记住，计算机不会自己思考，它仅仅是遵照指令办事。因此，为了构建一个可靠的操作系统，我们必须设计算法以克服各种可能出现的意外情况，不管它出现的概率有多小。

3.4.1　信号量

考虑一个分时/多任务操作系统，它控制只有一台打印机的计算机的活动。如果一个进程要求打印结果，那么它必须向操作系统提出请求，要求访问打印机设备的驱动程序。这个时候，操作系统必须根据该打印机是否被其他的进程占用来决定是否批准这个请求。如果没有被占用，那么操作系统应该批准这个请求，并允许该进程继续执行；否则，操作系统应当拒绝这个请求，也许把这个进程归类为等待进程，直到打印机可用为止。如果有两个进程同时获得对打印机的访问权，那么结果对两者都是不可取的。

微软的任务管理器

通过执行任务管理器这个应用程序，你可以对微软Windows操作系统的内部活动获得深刻的了解。（同时按下Ctrl、Alt和Delete键。）特别地，通过选择任务管理器窗口的进程标签，你可以看到进程表。在此，你可以体验一下：在激活任何应用程序之前，看一下进程表。（你也许会惊讶地发现表中已经有如此多的进程，它们对于系统的基本应用都是必不可少的。）现在激活一个应用，并且确认一个新进程已经进入到表中。你将还能够看到分配给进程的存储空间量。

为了控制对打印机的访问，操作系统必须跟踪打印机是否已经被分配。解决这个任务的一种方法是使用一个标志。在这里，它指存储器中的一个位，其状态通常是指置位（set）和清零（clear），而不是1和0。清零标志（值为0）表示打印机可用，置位标志（值为1）表示打印机当前已经分配出去了。表面上看，这种方法似乎可行。每次访问打印机的一个请求到来时，操作系统要做的工作仅仅是检查这个标志位。如果是清零标志位，那么操作系统就批准该请求，同

时将标志位进行置位。如果标志位已经置位，操作系统就将提出请求的进程放入等待队列中。每当一个进程完成了访问打印机的任务，操作系统要么将打印机分配给一个等待进程，要么在没有等待进程时，将这个标志清零。

然而，这个简单的标志系统还是有个问题。测试和可能有的标志置位任务也许需要几条机器指令。（从主存得到标志的值，在CPU中操控，最终写回主存。）因此，在检测到清零标志之后和标志被置位之前，这个任务有可能被中断。具体而言，假设这个打印机当前是可用的，且一个进程请求它的使用权，标志从主存中找到，而且发现它已清零，表示该打印机可用。但是，在这个时候这个进程被中断了，另一个进程开始了它的时间片，它也请求打印机的使用权。于是再一次从主存检索标志，仍发现它是清零的。因为前一个进程在操作系统要求从主存置位标志之前被中断了。因此，操作系统允许第二个进程使用打印机。过后，第一个进程在它被中断的地方恢复执行，那个地方正是操作系统发现标志是清零的地方。于是，操作系统继续对主存中的标志置位并允许第一个进程访问打印机。现在，这两个进程在使用同一台打印机。

这个问题的解决办法就是，要坚持让测试和可能有的标志置位任务必须在没有中断的条件下完成。一种方法是使用大多数机器语言都提供的中断屏蔽指令和中断允许指令。在执行时，中断屏蔽指令使未来的中断被锁定，而中断允许指令则使CPU恢复对中断信号的响应。于是，如果操作系统用中断屏蔽指令开始一个标志测试例程，并以中断允许指令结束，那么该例程一旦开始就不会有其他活动中断它。

另一种方法是使用**测试并置位**（test-and-set）指令，它在许多机器语言里可用。这条指令要求CPU检索一个标志的值，记住它，然后置位该标志，所有工作都在一条机器指令内完成。它的优点是，因为CPU在辨认一个中断之前必须完成当前的指令，所以测试任务和标志置位作为一条指令实现时不可能被分割。

刚才描述的一个正确实现的标志称为**信号量**（semaphore），它源自于控制轨道区段使用的铁路信号灯。事实上，信号量在软件系统里的用法与信号灯在铁路系统里的用法是一样的。就像一个轨道区间一次只能有一列列车，一段指令一次也只被一个进程执行。这样一段指令称为**临界区**（critical region）。一个临界区一次只允许被一个进程执行，这个要求称为**互斥**（mutual exclusion）。概括地说，获得对一个临界区的互斥的常用办法是用一个信号量守护这个临界区。一个进程要进这个临界区，必须确定这个信号量是清零的，并在进入临界区之前把它置位，然后在出临界区时把这个信号量清零。如果发现这个信号量在置位状态，那么试图进入临界区的进程必须等待，直到这个信号量被清零。

3.4.2 死锁

在资源分配中可能发生的另一个问题是**死锁**（deadlock）。在死锁状态下，两个或更多的进程被阻塞，不能执行，因为它们中的每一个都在等待已分配给另一个的资源。例如，一个进程可能已有对打印机的访问权，同时它还在等待访问计算机的CD播放器，而另一个进程有CD播放器的访问权，却在等待访问打印机。另一个例子出现在允许进程创建新的进程［这种活动在UNIX术语中称为**创建子进程**（forking）］来完成子任务的系统里。如果调度程序因为进程表没有空间而无法创建新的进程，同时系统里的每个进程又都必须创建额外的进程才能完成任务，那么没有一个进程可以继续。这种条件下（见图3-7）严重降低了系统的性能。

死锁状态的分析已经揭示，只有以下3个条件全部满足它才会出现。

（1）存在对不可共享资源的竞争。

（2）这些资源是在不完整的基础上请求的。也就是说，一个进程接受了某些资源后，稍后还将请求其他的资源。

（3）一个资源一旦被分配出去，就不能以强制的办法再收回。

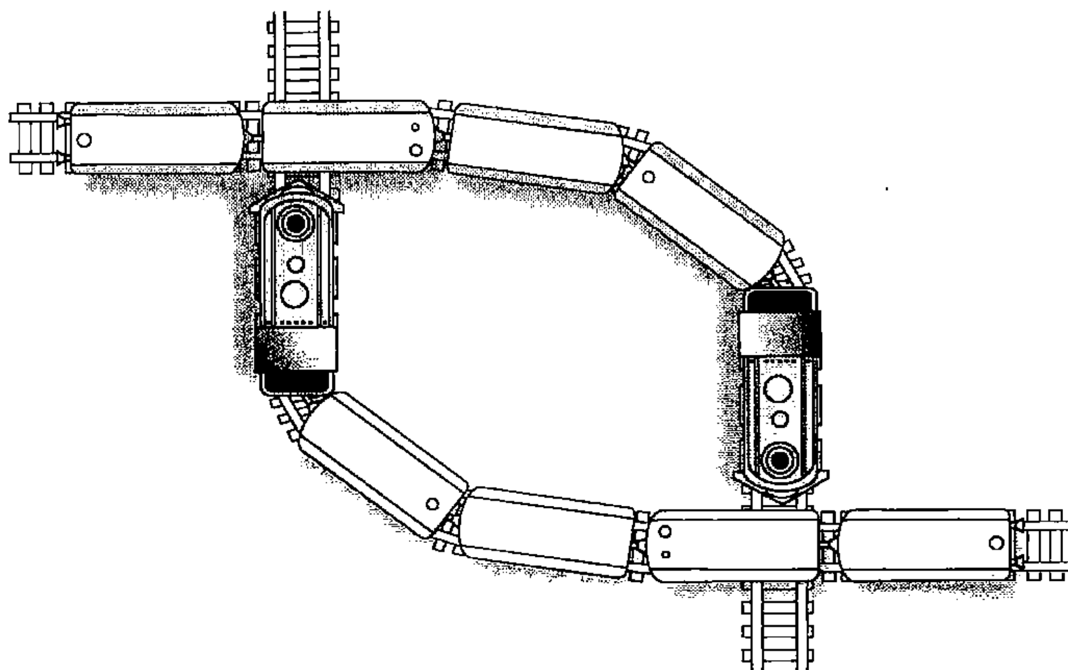


图3-7　由于竞争不可共享的铁路区间造成的死锁

分离出这些条件的意义在于，只要努力抑制这三个条件当中的任何一个，就可以避免死锁问题。一般认为，着力于抑制第三个条件的技术属于死锁检测和改正方案的范畴。在这些情况下，死锁状态被认为不大容易出现，因而不必特别采取办法避免死锁，而只是在死锁出现的时候检测出它，然后通过强制性收回某些已经分配出去的资源来改正它。进程表已满就属于这种情况。如果死锁是由于进程表满产生的，那么操作系统中的例程（或管理员利用其“超级用户”的特权）可以移除［专业术语是**清除**（kill）］某些进程，这将释放进程表的空间，打破死锁并使得剩下的进程可以继续它们的工作。

着力于抑制前两个条件的技术，一般被称为死锁避免方案。例如，针对第二个条件的一个方法是要求每个进程一次性请求它所需要的全部资源。另一个技术针对第一个条件，它不是直接地消除竞争，而是把不可共享的资源转变为可共享的资源。例如，假定出问题的资源是打印机，各种进程都请求使用它。每当一个进程请求打印机时，操作系统都会批准这个请求。但是，操作系统不是把这个进程连接到打印机的设备驱动程序上，而是连接到一个“虚构”的设备驱动程序上，该驱动程序把要打印的信息存放在海量存储器上，而不把它们发送到打印机上。于是，每个进程都认为它访问了打印机，所以能正常工作。以后，当打印机可用时，操作系统可以把数据从海量存储器传送到打印机。按照这个方法，操作系统通过建立多个虚构的打印机把不可共享的资源变成了好像是可共享的。这种保存数据供以后在合适的时候输出的技术称为**假脱机**（spooling）。

多核操作系统

传统分时/多任务系统通过以快过人类感知的速度快速切换时间片，制造了同时执行多个进程的假象。现代系统继续通过这种方式实现多任务处理，但最新的多核CPU确实能够同时运行2个、4个或更多个进程。与一组协同工作的单核计算机不同，一台多核机器包含多个独立的处理器（称为核），它们共享计算机的外围设备、内存等资源。对于一个多核操作系统，这意味着分派程序和调度程序必须考虑在每个核上应该执行哪些进程。随着不同的进程运行于不同的核上，进程间的处理竞争变得更具有挑战性，因为每当一个进程需要进入临界区时，所有核上都会禁用中断，但这种做法效率极低。构建能更好适用新的多核环境的操作系统机制，是计算机科学中比较活跃的研究方向。

假脱机是一种允许多个进程访问一个公共资源的技术，它可以有许多变体。例如，文件管理程序可以批准若干个进程访问同一个文件，前提是它们只是从该文件读取数据——如果多个进程试图同时更改一个文件就会发生冲突。于是，文件管理程序可以根据进程的需要分配对文件的访问权限，允许若干个进程有读访问权，但在任何给定时刻只有一个进程有写访问权。其他的系统可能把这种文件分成区段，使得不同的进程可以并发地更改文件的不同部分。然而，其中每一项技术要得到一个可靠的系统，都有一些枝节上的问题亟待解决。例如，当有写访问权的进程更改了这个文件，如何通知那些只有读访问权的进程呢？

问题与练习

1. 假定进程A和B共享同一台机器的时间,并且每个进程都需要短时间使用同一个不可共享的资源。（例如，每个进程可能都打印一系列独立的短报告。）每个进程都重复地获得这个资源，释放它，稍后又再次请求它。按照下面的方法控制对该资源的访问存在什么缺点？
   开始时，给一个标志赋予值0，如果进程A请求这个资源并且该标志为0，那么就批准这个请求，否则使进程A处于等待状态；如果进程B请求这个资源并且该标志为1，那么就批准这个请求，否则使进程B处于等待状态。每当进程A完成对这个资源的访问，就把标志变为1；每当进程B完成对这个资源的访问，就把标志变为0。
2. 假定一条双车道的道路在过隧道时合并为一个车道。为了协调这个隧道的使用，安装了下述信号系统：一辆汽车无论从哪个入口进入隧道，都会使隧道入口处上方的红灯点亮。当这辆汽车离开隧道时，红灯会灭。如果一辆到达的汽车发现红灯亮，那么它就要等待，直到红灯灭时方可进入隧道。
   这个系统存在什么问题？
3. 为了解决单行桥上两辆车相遇的死锁问题，假设已提出下面几个解决方案。说明每个解决方案各消除了前文中提到的3个死锁条件中的哪一个。
   a. 在桥上变空之前不允许汽车上桥。
   b. 如果两车相遇，让其中一辆车倒退。
   c. 在桥上加一个车道。
4. 假定我们用圆点表示多道程序设计系统中的每一个进程，从第一个圆点到第二个圆点的箭头表示第一个（圆点所表示的）进程等待（第二个圆点所表示）进程正在使用的（非共享）资源。数学家把得到的这种图称为**有向图**（directed graph），有向图的什么性质等价于操作系统的死锁问题？

3.5 安全性

由于操作系统管理着计算机的活动，很自然，它也在维护安全性方面起了重要的作用。从完整意义上说，安全性自身也有多种表现形式，可靠性就是其中一种。如果文件管理程序的缺陷使得一个文件的一部分丢失了，那么这个文件就是不安全的。如果一个分派程序里的缺陷导致系统故障（通常称为系统崩溃），使得一小时的打字工作白费了，那么我们会说，产品是不安全的。因此，计算机系统的安全性需要一个设计完美的可信赖的操作系统。

可靠软件的研发不再受制于操作系统，它贯穿整个软件开发过程。在计算机科学里称为软件工程，我们将在第7章讨论这个论题。在本节，我们集中讨论与操作系统息息相关的安全性问题。

3.5.1 来自机器外部的攻击

操作系统的一个重要任务就是，保护计算机的资源，防止受到非授权用户的访问。在不同的人使用计算机的时候，操作系统一般通过为不同的授权用户建立“账户”的方法来标记不同

权限的用户。账户实际上是包含了诸如用户的姓名、登录密码和用户的权限等条目的记录。操作系统在每个**登录**（login）过程中使用这些信息控制他们对系统的访问权限。（登录过程是一系列事务活动，在这个过程中，用户建立与计算机操作系统的初步联系。）

账户由**超级用户**（super user）或**管理员**（administrator）创建。在登录过程中通过了操作系统的管理员身份验证（通常是通过用户名和密码）的用户将享有很高的访问权限。这种联系一旦建立，管理员就可以更改操作系统的内部设置、修改关键的软件包、调整其他用户访问系统的权限，进行各种各样一般用户不能进行的活动。

通过这种“高级地位”，管理员能够监视计算机系统中的活动，检测到不管是恶意还是偶然的破坏行为。为了巩固这种关系，人们开发了大量称为**审计软件**（auditing software）的软件实用程序，来记录和分析发生在计算机系统中的活动。特别地，审计软件可以确定许多试图用错误的密码登录系统的活动，指示出非授权用户试图获得计算机访问权的行为。审计软件还可以识别用户账户中与该用户以往行为不一致的活动，这可能表明一个非授权用户访问了这个账户。（以下这样的事情不太可能发生：一个用户，以前仅仅使用文字处理和电子制表软件，然后突然开始访问系统的很专业的应用软件，或者试图执行超过其权限范围的实用软件包。）

设计审计软件的另外一个目的是检测**嗅探软件**（sniffing software），这种软件在计算机中运行时能够记录活动并稍后将之报告给潜在的入侵者。举一个老的但是很有名的例子，如果一个程序能够模拟操作系统的登录过程，那么这个程序就能被用来欺骗操作系统的授权用户，使他们认为自己是在和操作系统通信。然而，实际上他们是在和一个冒名顶替者通信，并在将自己的用户名和密码提供给冒名顶替者。

在所有与计算机安全相关的复杂技术问题上，让很多人感到吃惊的是，计算机系统安全领域中的主要难题之一就是用户自己的不小心。例如，用户选择的密码相对比较容易猜（如名字和生日等）；告诉朋友自己的密码；没有定时更换自己的密码；将自己的离线海量存储设备在机器间来回地转移，这样就潜在地降低了其安全性；在计算机系统中安装未经许可的软件，从而有可能损坏系统的安全性。对于上述问题，大多数使用大量计算机的机构都采用强制的策略，明文规定用户的需求和职责。

3.5.2　来自机器内部的攻击

一旦入侵者（也可能是怀有恶意的授权用户）获得了系统的访问权限，那么他们下一步的工作通常是浏览机器，寻找其感兴趣的信息或者是找地方插入破坏性软件。如果一个入侵者获取了系统的管理员账号，那么上述过程的发生就很自然了，这也是我们为什么要严格保护好管理员密码的原因。然而，如果是通过普通账号进行访问，那么入侵者必然会欺骗操作系统，以获得未授予该用户的权限。例如，入侵者会尝试着欺骗内存管理程序，让一个进程访问其被分配的存储区以外的内存区域；或者欺骗文件管理程序，访问本应该无权访问的文件。

今天，CPU在设计时已经加强了一些功能，能够阻止上面谈到的攻击尝试。举一个例子来说，我们可以考虑这样一个需求：通过内存管理程序，将进程限制在给它分配的内存区域内。如果没有这样的限制，一个进程就能够从内存中覆盖掉操作系统，从而接管对计算机的控制。考虑这样的一种威胁，为多道程序设计系统设计的CPU通常包括若干个专用寄存器，操作系统可以在这些寄存器中保存分配给一个进程的存储区域的上下界。于是，当执行该进程时，CPU把每个存储器引用与这些寄存器中的值进行比较，以保证该引用在指定的界限之内。如果发现这个引用在为该进程指定的区域之外，CPU将自动把控制权交还给操作系统（借助于中断处理），这样操作系统可以作出合理的处理。

这个方案中还存在一个小的但很重要的问题。如果没有进一步的安全措施，一个进程还是能够访问指定区域以外的内存单元，只要改变含有存储区界限的专用寄存器的值即可。也就是说，一个进程想要访问更多的内存区域，它只需要增加存放上界的寄存器的值，然后不需要得到操作系统的批准，就可以使用这些额外的内存区域。

为了防止这种恶意的活动，将多道程序设计系统的CPU设计为工作在两种**特权级**（privilege level）之一的模式下。我们将其中之一称为“有特权模式”，而另外一个称为“无特权模式”。当处在有特权模式下时，CPU能够用自己的机器语言处理所有的指令，但处在无特权模式下时，能够接受的指令就是有限的。这种仅在有特权模式下可用的指令，我们称为**特权指令**（privileged instruction）。（典型的有特权指令有：改变内存界限寄存器的内容的指令和改变CPU当前的特权模式的指令。）当CPU处于无特权模式时，任何执行特权指令的企图都将引起中断。这个中断将CPU转变为有特权模式，并将控制权交给操作系统内部的中断处理程序。

当开机时，CPU处于有特权模式，因此操作系统在引导过程后开始启动时，所有的指令都可以执行。然而，每当操作系统允许一个进程开始执行它的时间片时，就通过执行“改变特权模式”的指令，将CPU切换到无特权模式。于是，如果一个进程试图执行有特权指令，操作系统就会得到通知，这样操作系统就充当了维护计算机系统完整性的角色。

有特权指令和控制特权级别是操作系统维护安全性可用的一个主要工具。然而，使用这些工具对操作系统设计而言是一项复杂的任务，且在当前的操作系统中，错误还在不断出现。因此，在特权级别控制中，任何一点疏忽都可能给灾难打开大门，不论是恶意程序引起的，还是无意中的程序设计错误造成的。如果允许一个进程更改控制多道程序设计系统的计时器，那么这个进程就能够延长它自己的时间片，甚至控制整个机器。如果允许一个进程直接访问外围设备，那么它就能不受系统文件管理程序的监管而读取文件。如果允许一个进程访问分配给它的区域之外的内存单元，那么它就能访问甚至更改由其他进程正在使用的数据。因此，维护计算机的安全性，既是管理员的一个重要任务，也是操作系统设计的一个目标。

问题与练习

1. 列举几个密码选取不好的例子，并说明为什么不好。
2. 英特尔奔腾系列处理器提供4个特权级别，为什么CPU的设计人员选择提供4个特权级别，而不是3个或5个？
3. 如果多道程序设计系统里的一个进程可以访问分配给它的区域之外的存储单元，那么它怎样获得该机器的控制权？

复习题

（带*的题目涉及选读小节的内容。）

1. 列出一个典型操作系统的4个活动。
2. 概述批处理和交互式处理的区别。
3. 假设有3个作业R、S、T，按这个顺序排在一个作业队列里；接着，在第4个作业X进入队列之前，1个作业移出了队列，然后又有1个作业移出了队列，作业Y和作业Z排进队列，最后，按照一次一个作业地顺序移出，使队列变空。请按移出的顺序列出所有的作业。
4. 嵌入式系统和PC的差别是什么？
5. 什么是多任务操作系统？
6. 如果你有一台PC，列举它的几个多任务功能给你带来方便的情形。

7. 根据你所熟悉的计算机系统，列举两个应用软件组件和两个实用软件组件，然后说明为什么这样归类。
8. a. 操作系统的用户接口的作用是什么？
   b. 操作系统的内核的作用是什么？
9. 路径X/Y/Z描述的是什么目录结构？
10. 定义操作系统环境下使用的术语“进程”。
11. 操作系统的进程表里包含什么信息？
12. 就绪进程和等待进程的差别是什么？
13. 虚拟存储器和主存储器之间的差别是什么？
14. 假设某计算机有512 MB的主存，操作系统要创建主存两倍大小的页式虚拟内存，页面大小为2 KB，请问需要多少页？
15. 在分时/多任务系统里，如果两个进程同时访问同一个文件，会发生怎样混乱的情况？是否存在文件管理程序应批准这种请求的情形？是否存在文件管理程序应拒绝这种请求的情形？
16. 应用软件和系统软件之间的区别是什么？请各举一个例子。
17. 定义多处理器体系结构情况下的负载平衡与均分。
18. 概述引导过程。
19. 为什么说引导过程是必要的？
20. 如果你有一台PC，记录开机时你所观察到的活动序列。然后确定在引导进程实际开始工作之前有哪些信息显示在计算机屏幕上？什么软件写下这些信息？
21. 假定多道程序设计操作系统分配的时间片是10毫秒，计算机每纳秒平均执行5条指令，那么在一个时间片内能执行多少条指令？
22. 如果一个打字员每分钟能打60个单词（在这里假设一个单词含5个字符），问每打一个字符要多久？如果多道程序设计系统分配的时间片为10 ms，我们忽略进程间切换的时间，问打一个字符要分配多少时间片？
23. 假定一个多道程序设计系统分配时间片为50 ms，如果把磁盘的读写磁头定位到所希望的磁道上通常要花费8 ms，并且磁道上所要的数据旋转到读写磁头之下通常要17 ms，那么等待一个读磁盘操作发生可能要多少个时间片？如果该机器每纳秒能执行10条指令，那么在这个等待时间里可以执行多少条指令？（这就是为什么当一个进程用外围设备完成操作时，多道程序设计系统终止这个进程的时间片，让另一个进程运行而让第一个进程等待外围设备的服务。）
24. 列举一个多任务操作系统必须协调访问的5种资源。
25. 如果一个进程需要执行大量的I/O运算则称为I/O密集型进程，而如果一个进程由大多数在CPU/内存中完成的运算构成则称为计算密集型进程。如果这两种进程都在等待分配时间片，请问如何确定它们的优先级？为什么？
26. 在多道程序设计系统里运行两个进程，如果它们两个都是I/O密集型进程，或者一个是I/O密集型进程另一个是计算密集型进程（如上题所述），那么它们是否能达到较大吞吐量？为什么？
27. 编写一组指令告诉操作系统的分派程序，在一个时间片用完时该做什么？
28. 在进程状态中包含什么信息？
29. 列出多道程序设计系统中一个进程不会全部用完分配给它的时间片的情况。
30. 按照时间顺序列出一个进程被中断时发生的主要事件。
31. 按照你所使用的操作系统回答下列问题。
   a. 如何请求操作系统把一个文件从一个地方复制到另一个地方？
   b. 如何请求操作系统显示磁盘上的目录？
   c. 如何请求操作系统执行一个程序？
32. 按照你所使用的操作系统回答下列问题。
   a. 操作系统如何限制仅允许已批准用户访问资源？
   b. 如何让操作系统显示当前在进程表里的进程？
   c. 如何告诉操作系统你不想该机器的其他用户访问你的文件？

*33. 解释许多机器语言里“测试-置位”指令的重要用法。为什么整个测试-置位过程作为单个指令实现是重要的？

*34. 一个银行家只有100 000美元，贷款给两个客户，每位50 000美元。后来这两位客户回了同样的话：他们在能够还贷之前各自还需10 000美元，以完成与先前贷款有关的商业交易。这个银行家通过从其他地方借来资金来追加给这两个客户的贷款（提高贷款利率）解决了这个死锁问题。在死锁的三个条件中，银行家消除了其中的哪个条件？

*35. 每个想参加本地大学的铁路修建模型II课程的学生，都要得到教师的允许，并且交纳实验费。这两个要求可以在校园的不同地点办理，

可以按照任意顺序独立完成。注册学生限制为20名；这个限制由教师和财务处一起掌握，前者只授权20名学生，后者只收20名学生的费用。假定这个注册系统有19名学生成功注册了这一课程，但是最后这个名额有两名学生竞争——一名仅得到了老师的允许，另一名仅交纳了费用。下面解决该问题的各个方案中，分别消除了死锁的3个条件中的哪个？

a. 同意这两名学生都参加该课程。

b. 该班人数降为19人，因此这两名学生都不能注册该课程。

c. 拒绝这两名竞争的学生，让第三名学生作为第20名。

d. 注册该课程的要求改为一个：交纳费用。于是交了费用的学生注册成功，另一名被拒绝。

*36. 由于计算机显示屏每块区域一次只能被一个进程使用（否则屏幕中的图像将难以认清），这些由窗口管理程序分配的区域是不可共享的。为了避免死锁，窗口管理程序应消除死锁的3个必要条件中的哪一个？

*37. 假设一个计算机系统里不可共享资源分为3类：1级，2级，3级。其次，假设系统中的每一个进程都要根据这个类别请求它所需要的资源。也就是说，它请求2级资源之前必须一次请求所有必要的1级资源。一旦它得到了1级资源，就可以申请所有必要的2级资源，依次类推。这个系统会出现死锁吗？为什么？

*38. 机器人的两个手臂是程序控制的，它们从传送带上取零部件，测试它们的公差并根据结果分别把它们放到两个箱子中。零部件一次到达一个，它们之间有足够的距离。为了防止两个手臂尝试抓同一个零部件，控制手臂的计算机共享一个公共的存储单元。如果一个手臂在一个零部件到来时是可用的，那么控制它的计算机就读公共单元的值。如果该值非0，那么这一手臂让那个零部件通过，否则，起控制作用的计算机把一个非0的值放到这个存储单元，指挥那个手臂抓起该零件，动作完成后再把值0存入该存储单元。什么样的事件序列可能导致两个手臂之间激烈竞争？

*39. 说明队列在假脱机输出到打印机的过程中的使用。

*40. 如果一个等待时间片的进程一直都没有获得时间片，这称为**饥饿**（starvation）。

a. 对于竞相通过十字路口的汽车来说，十字路口的地面是不可共享的资源。控制这个资源分配的是红绿灯，不是操作系统。如果这个灯能够感知来自每个方向的交通流量，并通过程序给较大流量的方向以绿灯，流量少的方向就可能饥饿。请问“饥饿”现象怎么避免？

b. 在一个进程优先级保持固定的优先级系统中，如果调度程序总是按优先级分配时间片，那么在什么时候一个进程会感到“饥饿”？（提示：相对于正在等待的进程来说，刚执行完时间片的进程的优先级是多少，并且接下来按哪种规则分配下一个时间片？）你能猜到各种操作系统是怎么避免这个问题的吗？

*41. 死锁和饥饿（参见习题40）的相似之处是什么？差别又是什么？

*42. 下面是“哲学家进餐”问题，它最初是由狄杰斯特拉提出的，现在已经是计算机科学传说中的一部分。

5个哲学家围着一个圆桌就座，每个人面前放了一盘细面条。桌上有5把叉子，每个盘之间有一把，每个哲学家都在思考和吃面之间轮换。为了吃面，一个哲学家需要拥有紧挨他盘子的2把叉子。

说明“哲学家进餐”问题中的死锁和饥饿（参见习题40）问题。

*43. 对于一个多道程序设计系统中的时间片，如果使其越来越短，那么会发生什么情况？越来越长呢？

*44. 随着计算机科学的发展，机器语言已被扩展以提供专门指令。在3.4节中介绍了这样3条在操作系统中广泛使用的指令。这些指令是什么？

45. 列举操作系统管理员能执行而一般用户不能执行的两个活动。

46. 操作系统如何防止一个进程访问另一个进程的存储空间？

47. 假定一个口令由9个取自英文字母表（26个字符）的字符组成。如果测试每个可能的口令需要1毫秒，那么测试所有可能的口令需要多长时间？

48. 为什么为多任务操作系统设计的各个CPU能够在不同特权级运行？

49. 列出两个由有特权的指令请求的典型活动？

50. 列举一个进程可能挑战计算机系统（如果未被操作系统防止这样做）安全性的3种方式。

51. 什么是多核操作系统？
52. 固件更新和操作系统更新之间的区别是什么？
53. 窗口管理程序与操作系统有什么关系？
54. 因特网浏览器是微软的Windows操作系统的一部分吗？
55. 嵌入式操作系统能够解决哪些特殊问题？

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 假定你在使用一个多用户操作系统，它允许你查看其他用户的文件的名字，而且如果其他用户的文件没有加保护措施，它还允许查看那些文件的内容。未经允许就查看这些信息类似于未经允许就闲逛别人未锁门的房间，还是类似于阅读放在公共休息室（如医生的候诊室）的资料？
2. 若访问一个多用户计算机系统，你在选择口令时有什么责任？
3. 如果一个操作系统有安全性缺陷，使得一个恶意的程序员能够在未经批准的情况下访问其敏感数据，那么该操作系统的开发人员应该负多大的责任？
4. 你有责任锁好门防止入侵者入内，还是公众有责任在未受邀请时待在门外？操作系统有责任防备别人对计算机及其内容的访问，还是非法入侵者有责任不入侵计算机？
5. 在《瓦尔登湖》一书中，梭罗坚持认为，我们已经变成自己工具的工具。也就是说，我们并非从所拥有的工具中受益，而是要花费时间得到工具和维护工具。对于计算机，这在多大程度上是真的？如果你有一台个人计算机，那么你花多少时间去赚钱承担它的费用、学习如何使用它的操作系统、学习如何使用它的实用程序和应用软件、维护它，以及为它的软件下载更新包？你得到的好处的时间量与你花费的时间总量相比又如何？使用它时，你花费的时间值得吗？有或没有个人计算机会对你的人际交往活跃度有影响吗？

课外阅读

Bishop, M. *Introduction to Computer Security*. Boston, MA: Addison-Wesley, 2005.

Davis, W. S. and T. M. Rajkumar. *Operating Systems: A Systematic View*, 6th ed. Boston, MA: Addison-Wesley, 2005.

Deitel, H. M., P. J. Deitel, and D. R. Choffnes. *Operating Systems*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2005.

Nutt, G. *Operating Systems*: *A Modern Approach*, 3rd ed. Boston, MA: Addison-Wesley, 2005.

Rosenoer, J. *Cyberlaw, The Law of the Internet*. New York: Springer, 1997.

Silberschatz, A., P. B. Galvin, and G. Gagne. *Operating System Concepts*, 8th ed. New York: Wiley, 2008.

Stallings,W. *Operating Systems*, 5th ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Tanenbaum, A. S. *Modern Operating Systems*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2008.

第4章

组网及因特网

本章讨论计算机科学中称为组网的领域，包括学习如何将计算机连接起来共享信息和资源。学习的内容包括网络的结构与操作、网络的应用以及网络安全问题。学习的一个重点主题是遍布世界范围的特殊网络——因特网。

本章内容

人们对不同计算机之间共享信息和资源的需求催生了相互连接的计算机系统，它被称为**网络**（network）。计算机通过网络连接在一起，数据可以从一台计算机传输到另一台计算机。在网络中，计算机用户可以相互交换信息，并且可以共享分布在整个网络系统中的资源，如打印功能、软件包以及数据存储设备。用于支持这类应用的基础软件也已经从简单的实用软件包升级为扩展网络软件系统，从而可以提供一个复杂的网络范围的基础架构。从某种意义上说，网络软件正发展成为网络范围的操作系统。本章将探讨计算机科学中这个不断发展的领域。

4.1 网络基础

我们从多种基本的网络概念开始学习网络。

4.1.1 网络分类

计算机网络通常分为LAN（Local Area Network，局域网）、MAN（Metropolitan Area Network，城域网）和WAN（Wide Area Network，广域网）。局域网通常由一幢建筑物或者综合建筑楼群中的若干计算机组成。例如，大学校园的计算机或者工厂中的计算机都可以用局域网连接。城域网属于中型网络，例如可以覆盖某一社区。广域网连接的计算机覆盖范围更广，可以是相邻的城市，也可以是在世界的两端。

网络的另一种分类依据是，网络的内部操作是基于公共领域的设计，还是基于特定实体（如个人或公司）所拥有并控制的创新。前一种类型的网络称为**开放式**（open）网络，后一种称为**封闭式**（closed）网络，有时也称为**专用**（proprietary）网络。开放式网络允许自由流通，因此更容易被大众所接受，这就是它们通常最终战胜专有网络的原因，因为专有网络的应用受到许可费和合约条件的限制。

因特网（马上要讲到的一种全球流行的网络的网络）属于开放式系统。尤其是，贯穿因特网的通信是由一组称为TCP/IP协议簇的开放标准控制的（见4.4节）。任何人都可以自由地使用这些标准，而不需要付费或是签署许可协议。相反，像Novell这样的公司可能开发一些存在所有权的系统，并通过出售或出租它们获利。

网络的另一种分类方法依据的是网络拓扑学，网络拓扑是指计算机的连接模式。众多拓扑中，比较常见的两种类型是：总线型拓扑，即所有计算机都通过同一条被称为“总线”的通信线路连接起来（如图4-1a）；星型拓扑，即将一台计算机作为中心，所有其他计算机都与之相连（如图4-1b）。20世纪90年代，总线型拓扑得以流行，当时是通过称为以太网的一组标准实现的，而且以太网依然是目前使用最广泛的组网系统之一。星型拓扑可以追溯到20世纪70年代，它由一台大型中央计算机服务于多个用户的范例发展而来。随着用户使用的简单终端发展成为小型计算机，星型拓扑应运而生。目前，星型拓扑配置在无线网络中应用比较广泛，无线网络利用无线电广播和中央计算机实现通信，这里的中央计算机称为AP（Access Point，接入点），是协调所有通信的焦点。

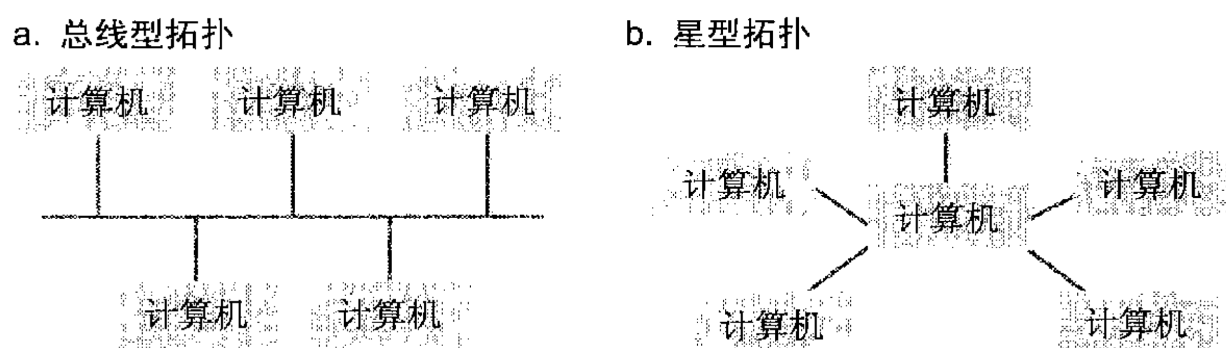


图4-1　两种常见的网络拓扑结构·

以太网

以太网是一组标准，用于实现利用总线拓扑结构的局域网。它的名字源于最初的以太网设计，其中的计算机由称为以太的同轴电缆连接。以太网在20世纪70年代开始开发，现在已经由IEEE标准化，成为IEEE 802标准体系的一部分。以太网是个人计算机组网的最通用方法。的确，用于PC的以太网卡现已成为标准组件，可在目前的零售市场上买到。

事实上，今天有很多版本的以太网，反映了技术的进步以及更高的传输速率。然而，各种版本都具有以太家族的公共特性，其中包括将传输用的数据打包的格式，实际传输的二进制位曼彻斯特编码（表示0和1的一种方法，0表示为向下跳变的信号，1表示为向上跳变的信号）的使用，以及使用CSMA/CD控制传输的权限。

总线型网络和星型网络在计算机物理排列上的区别并非总是很明显。二者的区别在于网络中的计算机是通过一条公共总线直接互相通信，还是通过中央计算机媒介间接通信。例如，总线型网络可能不会出现一条长总线且与计算机的连线却很短的情况，如图4-1所示。相反，总线型网络可能拥有一条非常短的总线，而与每台计算机的连线却很长，这意味着总线型网络看起来会比较像星型网络。确实，有时需要将每台计算机与中央位置通过网线连接，而在中央位置又把它们连接到一种叫做**集线器**（hub）的设备，从而构成总线网络。集线器其实就是一条非常短的总线，其功能在于将接收到的任何信号（可能会经过一些放大）传回给与之相连的所有计算机。尽管其结果看起来像星型网络，但操作上像总线型网络。

4.1.2　协议

为了网络运行可靠，必须建立管理网络活动的规则，这类规则称为**协议**（protocol）。通过

开发及采纳协议标准，商家生产的网络应用产品则可以与其他商家的产品兼容。因此，在网络技术的开发中，协议标准的开发是一个必不可少的环节。

为了解协议概念，我们考虑这样一个问题，如何协调网络中计算机之间报文的传输。如果没有控制此类通信的规则，所有的计算机就很可能同时抢着传输报文，或者在其他机器需要协助时也无法提供帮助。

在基于以太网标准的总线型网络中，报文传输的许可是通过名为CSMA/CD（Carrier Sense, Multi-Access with Collision Detection，带冲突检测的载波侦听多路访问）的网络协议控制的。该网络协议规定每条报文都要广播给总线上的所有计算机（如图4-2所示）。每台计算机都对所有报文进行监听，但是只保存发送给自己的报文。为了传输报文，计算机需要等到总线处于空闲状态，然后才开始传输报文并继续监听总线。如果另一台计算机也开始传输报文，那么两台计算机都会检测到这种冲突，并各自暂停一段随机长的时间，然后再次尝试传输。这种结果就像一小组人在交谈中使用的次序一样，如果两个人同时开始讲话，他们两个都会停下来。不同的是，人们可能会有一系列对话，如："对不起，刚才你想说什么？""不，不，你先说。"但是在CSMA/CD网络协议下，每台计算机都只会稍后再作尝试。

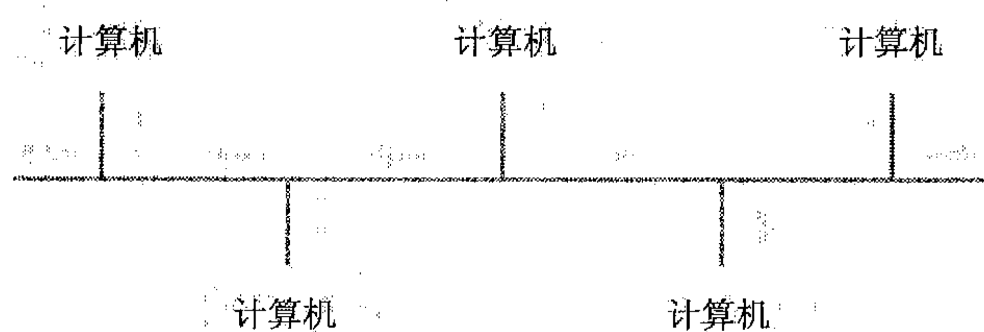


图4-2 总线型网络的通信

注意，CSMA/CD和无线星型网络并不兼容。在无线星型网络中，所有的计算机都通过中央接入点通信，原因在于一台计算机可能无法检测到与其他计算机的传输冲突。例如，一台计算机可能监听不到其他计算机，因为自己的信号淹没了其他计算机的信号。另一个原因可能是不同计算机传输的信号由于障碍物或者距离的原因互相阻塞，虽然它们都能与中央接入点通信，这种情况称为**隐藏终端问题**（hidden terminal problem），如图4-3所示。这使得无线网络采用避免传输冲突的方法，而不是检测冲突的方法。这种方法被归类为CSMA/CA（Carrier Sense, Multiple Access with Collision Avoidance，带冲突避免的载波侦听多路访问），其中很多方法是由IEEE（参见第7章中的"美国电气及电子工程师协会"）在IEEE 802.11中定义的协议下进行标准化的，通常被称为WiFi（无线保真）。需要强调的是，冲突避免协议的设计目的是避免冲突，也许并不能完全消除冲突。当冲突发生时，必须重新传输消息。

冲突避免最常见的方法是将优先权赋予已经在等待传输机会的计算机。这种协议和以太网的CSMA/CD有相似之处。二者的主要区别在于当一台计算机首先需要传输报文，并且发现信道处于空闲状态时，它并不是立即开始传输。相反，它会等待短暂的时间，只有当信道在这一段时间内都保持在空闲状态时才会开始传输。如果在这个过程中，遇到信道被占用的情况，那么计算机就会等待一段时间，时间的长度随机决定，然后再重新尝试传输。一旦这段时间耗尽，计算机就被允许立即占用空闲的信道。这就意味着避免了"新来者"和已经处于等待状态的计算机之间的冲突，因为"新来者"需要等到一直处于等待状态的计算机开始传输之后，才会被允许占用空闲的信道。

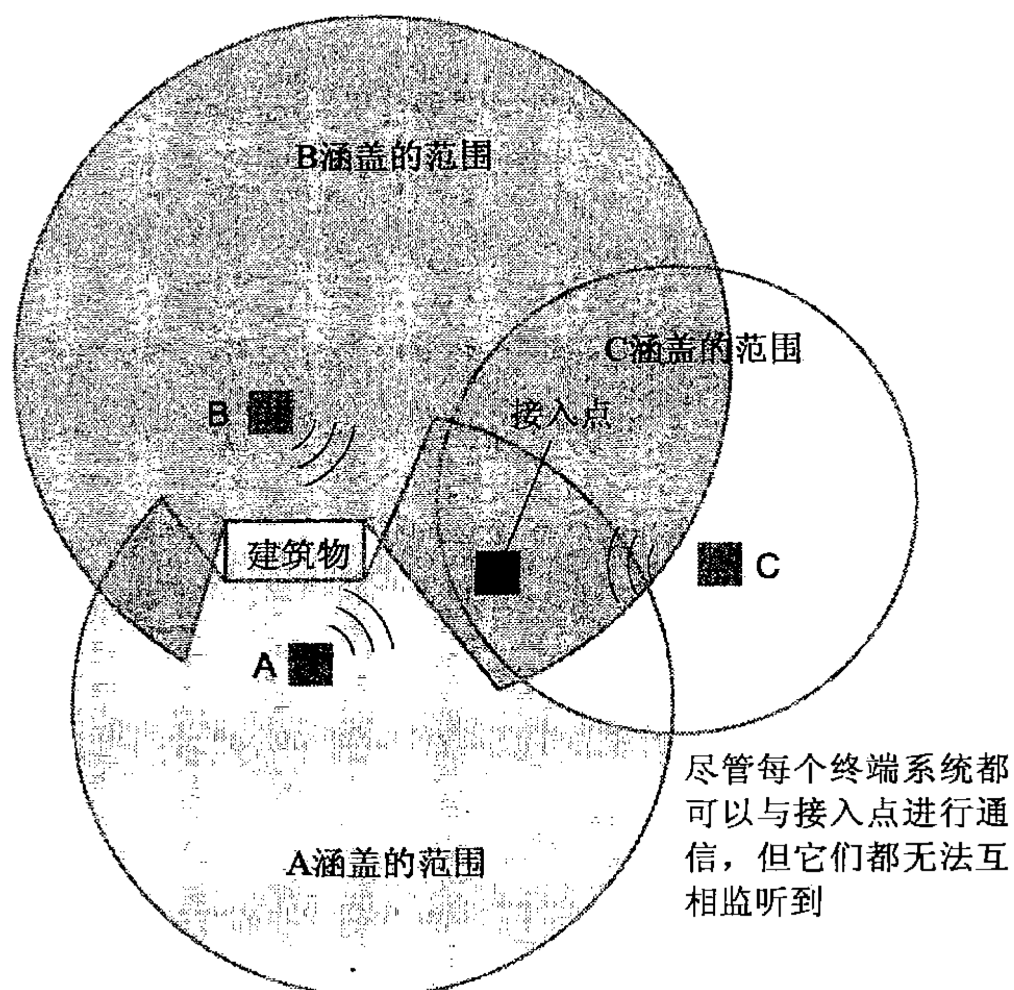


图4-3　隐藏终端问题

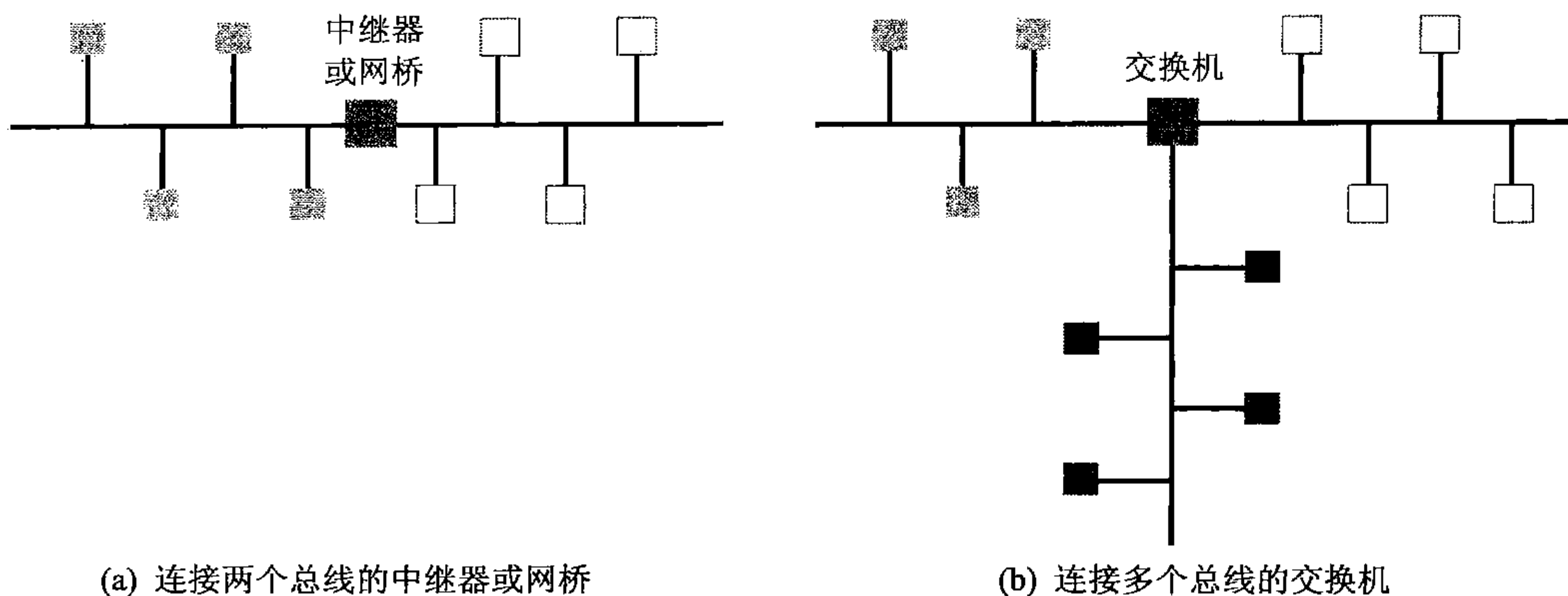
然而，该协议无法解决隐藏终端问题。毕竟，任何基于辨别空闲或者繁忙信道的协议都需要每个站点能够监听到其他所有站点。为了解决这一问题，一些WiFi网络要求每台计算机向接入点发送简短的“请求”报文，并等待接入点确认收到请求，然后再传输完整的报文。如果接入点由于正在处理“隐藏终端”处于繁忙状态，将会忽略请求，然后请求的计算机将获悉并等待。否则，接入点会确认请求，并且计算机将获悉现在进行传输是安全的。注意，尽管计算机无法监听到正在发生的报文传输，但是网络中所有的计算机都能监听到接入点发出的确认，因此就能知道接入点在任何特定的时间是否繁忙。

4.1.3　网络互连

有时候需要连接现存的网络以形成一个扩展的通信系统，形成相同类型的更大的网络。例如，对于基于以太网协议的总线型网络，经常可以将总线连接起来以形成一个长总线。它是利用中继器、网桥、交换机等不同的设备完成的，这些设备区别微妙且信息量大。最简单的是**中继器**（repeater），它仅仅是在两个原始总线间简单地来回传送信号（通常有某种形式的放大）的设备，而不会考虑信号的含义（如图4-4a所示）。

网桥（bridge）类似于中继器，但是更复杂一些。它也是连接两条总线，但是不必在线路上传输所有的报文。相反，网桥要检查每条报文的目的地址，并且当该报文的目的地是另一边的计算机时才将其在线路上传输。因此，在网桥同一侧的两台计算机不需要打扰另一边的通信就可以互相传输报文。相对于中继器，网桥形成的系统更加高效。

交换机（switch）本质上就是具有多连接的网桥，可以连接若干条总线，不止两条。因此，交换机形成的网络包括若干从交换机延伸出来的总线，它们就类似于车轮的辐条（图4-4b）。与网桥一样，交换机也要考虑所有报文的目的地址，并且仅仅转发那些目的地是其他“辐条”的报文。此外，被转发的报文只会送至相应的“辐条”，因此减轻了每根“辐条”的传输流量。



(a) 连接两个总线的中继器或网桥　　(b) 连接多个总线的交换机

图4-4 将小型总线网组建成大型总线网

需要注意的是，当计算机通过中继器、网桥以及交换机连接时得到的是一个大网络。整个系统用相同的方式运作（使用相同的协议），就像每个规模较小的初始网络一样。

然而，要连接的网络有时候会有不兼容的特性。例如，WiFi网络的特性就可能与以太网网络中的不兼容。在这种情况下，网络必须按建立一个网络的网络［称为**互联网**（internet）］方式连接。在这个网络中，原始的网络仍然保持其独立性，并且继续作为独立的网络运行。（注意，普通名词互联网（internet）不同于因特网（Internet）。后者首字母要大写，指的是一种独特的、世界范围的互联网，在本章的其他节会学到。现实中有许多互联网的例子。事实上，在因特网流行之前，传统的电话通信就是由世界范围的互联网系统操控的。）

把网络连结起来形成互联网的设备是**路由器**（router），这是一种用来传送报文的专用计算机。注意，路由器的任务与中继器、网桥和交换机的不同，路由器提供了网络之间的链接，并允许每个网络保持它独特的内部特性。作为一个例子，图4-5描述了通过路由器组连接两个WiFi星型网络和一个以太网总线网络的情形。当某个WiFi网络中的一台计算机想要给以太网中的一台计算机发送报文时，它首先会把报文发送到其网络中的接入点，接入点再把报文发送到与之相连的路由器，该路由器把报文转发至以太网中的路由器。在那里该报文被发送给总线上的一台计算机，然后这台计算机把报文转发到它在以太网中的最终目的地。

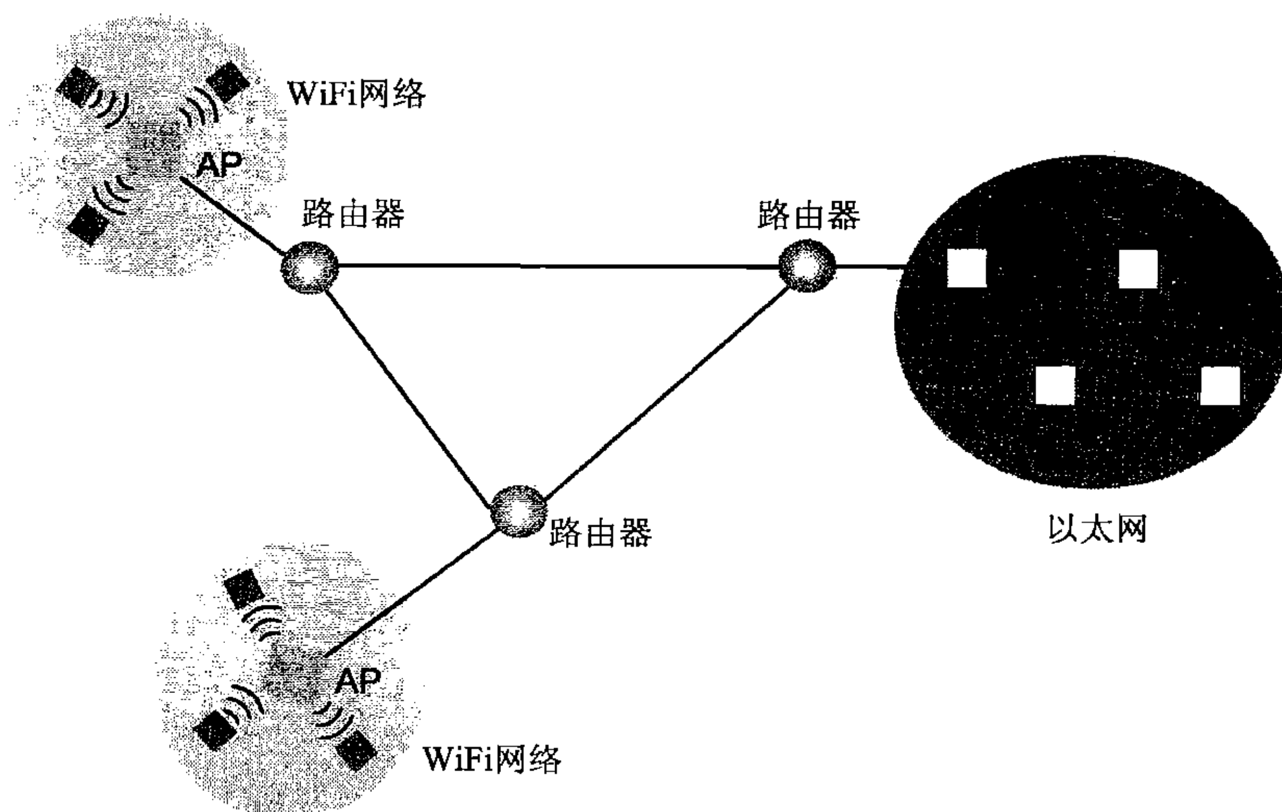


图4-5 路由器连结了两个WiFi网络和一个以太网络，形成了一个互联网

路由器得名的原因在于它的用途是向适当的方向转发报文。这个转发过程基于互联网范围的寻址系统，其中互联网上的所有设备（包括原始的网络中的计算机和路由器）都被赋予了唯一的地址。（这样，原始网络中的每台计算机都有两个地址：自己网络内的“本地”地址和它的互联网地址。）一台计算机想给远程网络中的另一台计算机发送报文，就要附上报文的目的地的互联网地址，然后把报文发送给其本地的路由器，在那里报文将向正确的方向转发。为了转发报文，每个路由器都维护了一张**转发表**（forwarding table)，该表中包含了根据目的地地址消息应该发送的方向这种路由信息。

一个网络与互联网链接的“点”经常称为**网关**（gateway），因为它是网络与外部世界之间的通道。网关有多种形式，因而这个术语的使用不太严谨。在许多情况下，网络的网关仅仅是路由器，通过它网络能与互联网上的其他网络通信。在其他情况下，术语网关所指的可能不仅仅是一台路由器。例如，在连接到因特网的多数住宅WiFi网络中，术语网关指的是网络的接入点和与接入点相连的路由器，因为这两个设备通常安装在一个单元中。

4.1.4　进程间通信的方法

一个网络中，在不同计算机上执行（甚至在一台计算机上通过分时/多任务处理执行）的各种活动（或进程）必须经常互相通信，以便协调行动并完成指派的任务。这种进程之间的通信称为**进程间通信**（interprocess communication)。

进程间通信通常采用的是**客户机/服务器**（client/server）模型。这种模型规定了进程的基本角色，或者是向其他进程发出请求的**客户机**（client)，或者是满足客户机请求的**服务器**（server)。

客户机/服务器模型最初应用于连接办公室间的所有计算机的网络。这样，网络中的所有计算机都可以使用连接到该网络上的性能良好的唯一打印机，在这种情况下，打印机的角色就是服务器（常称为**打印服务器**，print server)，其他计算机则为客户机，传递打印请求给打印服务器。

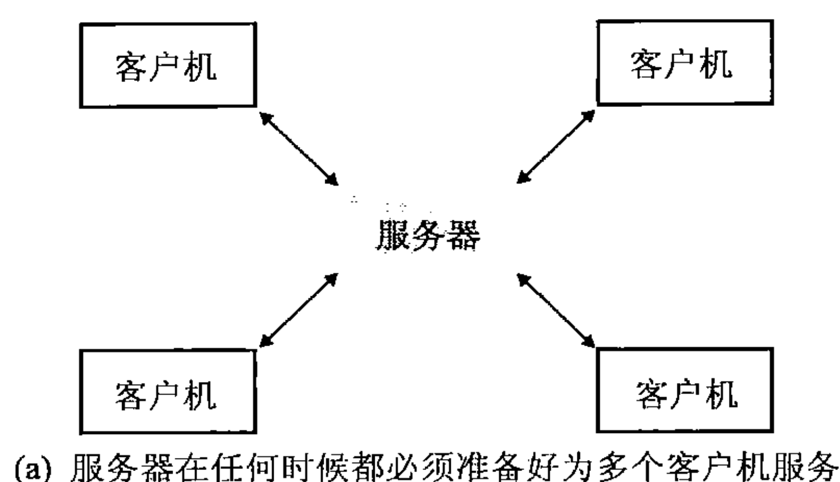
客户机/服务器模型的另外一种早期应用，用于减少磁盘存储开销以及复制记录的需要。在这种情况下，网络中的某一台计算机需要配置高容量海量存储系统（通常是磁盘），存储某一组织的所有记录，那么网络中其他计算机就可以在需要这些记录时提出请求。于是，包含记录的计算机的角色就是服务器，称为**文件服务器**（file server)，而那些向文件服务器提出存取请求的计算机就扮演客户机的角色。

客户机/服务器模型在当今的网络应用中使用广泛，在本章后面几节会介绍这一点。不过，客户机/服务器模型不是进程间通信的唯一方式，另外一种是**对等**(peer-to-peer，通常简称为P2P）模型。客户机/服务器模型为一个进程（服务器）与另外多个进程（客户机）通信，而对等模型为两个进程间对等通信（见图4-6)。此外，服务器必须持续运作，以便随时服务于客户机，但是对等模型涉及的则是临时执行的进程。例如，对等模型的应用包括发送即时消息（其中人们通过因特网进行文字交流），也可以是人们玩交互式竞技游戏。

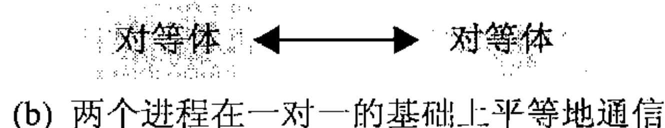
对等模型也是分发文件（如因特网上的音乐和电影）的常用方法。在这种情况下，一个对等体可以从另一个对等体接收文件，然后把此文件提供给其他的对等体。以这种方式参与分发的对等体集合有时称为蜂群（swarm)。文件分发的蜂群方法与先前使用客户机/服务器模型的方法相反，该模型需要建立中央分发中心（服务器），以使客户端从该中心下载文件（或至少是找到那些文件的源）。

从文件共享的角度来看，P2P模型正在取代客户机/服务器模型的一个原因在于它把服务分布到许多的对等体上，而不是集中在一个服务器上。这种非集中化的操作构建了更高效的系统。遗憾的是，基于P2P模型的文件分发系统流行的另一个原因在于（假设合法性遭到质疑）缺乏

中心服务器使得版权执法行动变得更为困难。但是，也存在许多案例，其中一些人发现“困难”并不意味着“不可能”，并且由于对版权的侵犯，他们发现自己面临着重大的责任。



(a) 服务器在任何时候都必须准备好为多个客户机服务



(b) 两个进程在一对一的基础上平等地通信

图4-6 客户机/服务器模型与对等模型的比较

你可能经常读到或者听到“对等网络”这个说法，这个例子正好说明了，非科技界采用科技术语时是如何产生误用的。“对等”指的是两个进程通过网络（或者互联网）通信的一种系统，并不是网络（或者互联网）的一种特性。一个进程可以开始采用对等模型与另外一个进程通信，接着又采用客户机/服务器模型通过同一个网络与另外的进程通信。因此，精确地说应该是利用对等模型通信，而不是通过对等网络通信。

4.1.5 分布式系统

因为组网技术的成功，计算机之间通过网络的交互已经很普遍，并且涉及方方面面。许多现代软件系统，例如全球信息检索系统、公司范围的会计和库存系统、计算机游戏甚至操控网络基础设施本身的软件，都被设计成**分布式系统**（distributed system）。这意味着，它们由在网络中不同计算机上作为进程执行的软件单元组成。

分布式系统最开始是独立开发的。不过，今天的研究已经发现在这些系统之间可以使用公共的基础设施，例如通信系统以及安全系统。于是，人们开始努力生产能够提供这种基础设施的预制系统。因此要构建一个分布式应用，只需要开发系统中应用所特有的部分即可。

现在，一些类型的分布式计算系统已经很常见。**集群计算**（cluster computing）指的是一种分布式系统，其中多个独立的计算机密切合作，提供的计算和服务可与大得多的机器相比拟。这些独立的机器，加上连接它们的高速网络的总成本，要比一台高价位的超级计算机的成本低，但其可靠性更高，且维护成本更低。这样的分布式系统用于提供**高可用性**（high-availability，因为这更有可能保证集群中至少有一台计算机可以响应请求，即使集群中其他计算机发生故障或不可用）和**负载平衡**（load-balancing，因为负载可以自动从集群中负载太大的计算机转移到负载很小的计算机上）。**网格计算**（grid computing）是指与集群相比耦合度不高的分布式系统，但其成员机器仍共同协作来完成大型任务。网格计算可能需要专门的软件，以便更容易地发布数据和算法到网格中的计算机。威斯康星大学的Condor系统和BOINC（Berkeley's Open Infrastructure for Network Computing，伯克利开放式网络计算平台）都属于这种类型。这两种系统通常安装在用于其他目的的计算机上（如办公用PC和家用PC），当机器空闲时便可为网格自愿提供计算能力。由于因特网日益增强的连通性，即这种自愿行动，分布式网格计算使得数以百万计的家用PC都可以解决无比复杂的数学问题和科学问题。凭借**云计算**（cloud computing），

网络上大量的共享计算机得以被按需分配给客户机使用。云计算是分布式系统中的最新趋势。20世纪初大城市电网的发展使得个体工厂和企业不再需要维护自己的发电机，与此非常类似，因特网使得实体可以将其数据和计算委托给“云”，这里的“云”指的是网络上可用的大量计算资源。例如，亚马逊的弹性计算云（Elastic Compute Cloud）服务允许客户按小时租用虚拟计算机，而不需要考虑计算机硬件的实际位置。另外，Google Docs和Google Apps允许用户在信息方面进行协作，允许他们在构建Web服务时不需要了解多少台计算机在用于求解同一问题或相关数据存储在何处。云计算服务就可靠性和可扩展性提供了合理保证，但却引发了人们对于隐私和安全性的担忧，因为我们也许再无法知晓谁在拥有着、操作着我们所用的计算机。

问题与练习

1. 什么是开放式网络？
2. 归纳网桥和交换机之间的区别。
3. 什么是路由器？
4. 列举社会中一些符合客户机／服务器模型的关系。
5. 列举社会上应用的一些协议。
6. 请概述集群计算和网格计算之间的区别。

4.2　因特网

互联网中最著名的例子就是**因特网**（Internet，注意首字母是大写的），它起源于20世纪60年代的研究项目。它的目的是开发出将许多计算机网络链接起来的能力，以便它们能作为一个连接的系统发挥作用，而这个系统不会由于局部灾难而瓦解。这个项目的工作绝大部分是由美国政府资助并通过DARPA（Defense Advanced Research Projects Agency，美国国防部高级研究计划署）发起的。经过这么多年，因特网的开发已经从政府资助的项目转变成了学术研究项目，而且如今在很大程度上，它已经是商业项目了，用于连接全世界局域网、城域网及广域网，涉及上百万台计算机。

第二代因特网

既然因特网已经由科研项目转变成一种日常产品，科研界于是转向了称为第二代因特网（Internet2）的项目。他们打算将其定位于学术系统，涉及许多与工业及政府有合作关系的大学。目标是在需要高带宽的因特网应用中进行科研，包括远程访问及控制昂贵的最新型设备，例如望远镜和医学诊断仪器。现在的一个科研例子是：由机器人实施远程外科手术，机器人的手模仿远程外科医生的手，而外科医生通过视频观察病人。你可以通过网站http://www.internet2.org了解到更多关于第二代因特网的信息。

4.2.1　因特网体系结构

我们已经提到，因特网是相连网络的集合。总体上，这些网络的构建和维护是由ISP（Internet Service Provider，因特网服务提供商）来完成的。就网络本身而言，通常也习惯使用术语ISP来表示。因此，当说到要连接到一个ISP时，我们真正的意思是连接到由ISP所提供的网络。

由ISP运作的网络系统可以按照各网络在整个因特网结构中所起的作用分类成层次结构（如

图4-7所示)。整个层次结构的顶部是数量相对较少的**第一层ISP**(tier-1 ISP),这些ISP拥有非常高速、高容量的国际化广域网。这些网络被看成是因特网的主干,它们通常是由通信行业中的大公司来运作的。例如,有一家公司最初是传统的电话公司,后来它们把服务领域扩展到提供其他通信服务。

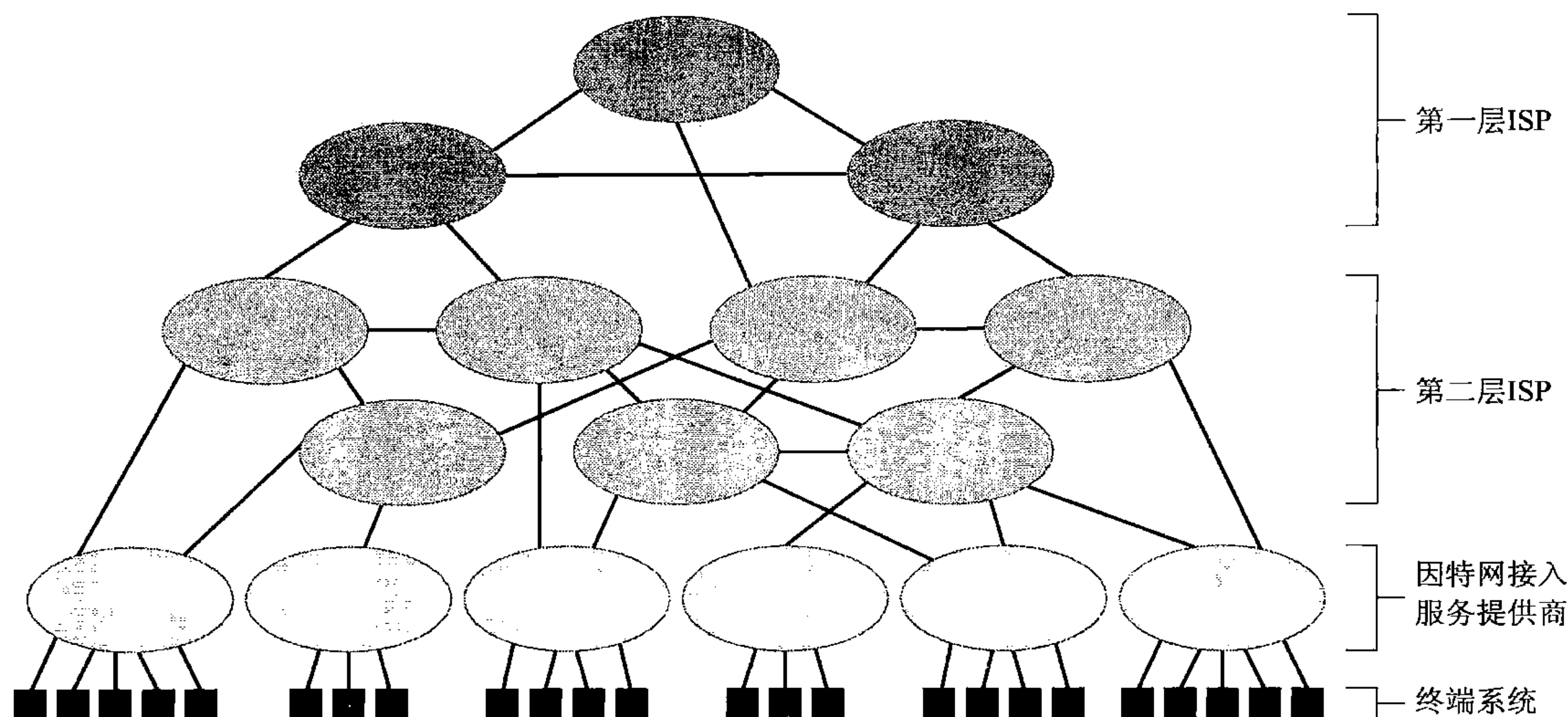


图4-7 因特网的构成

与第一层ISP连接的是**第二层ISP**(tier-2 ISP),第二层因特网服务提供商往往是区域性的,在实力上也稍逊一些。(第一层和第二层因特网服务提供商的区别通常是仁者见仁,智者见智。)此外,这些网络通常也是由通信行业的公司运营。

第一层和第二层因特网服务提供商本质上是路由器的网络,集中提供因特网通信基础设施。它们同样可以被认为是因特网的核心。通常由称为**因特网接入服务提供商**(access ISP)的中间商提供与这一核心的接入服务。因特网接入服务提供商本质上是独立的互联网,有时也称为**内联网**(intranet),由向个人用户提供因特网接入业务的机构来运营。这些公司包括AOL、微软以及本地通过提供服务收取服务费的有线电视和电话公司,另外还包括一些大学或者公司等组织,它们为组织内部的个人用户提供因特网接入。

个人用户与因特网接入服务提供商连接的设备称为**终端系统**(end system)或者**主机**(host)。这些终端系统并不一定是传统意义上的计算机。它们包括很多种设备,如电话、摄像机、汽车和家用电器。毕竟,因特网本质上是一个通信系统,因此任何可以与其他设备进行通信并从中获益的设备都是潜在的终端系统。

终端系统与因特网接入服务提供商连接的技术也不尽相同。或许,发展最快的是基于WiFi技术的无线连接。策略是将接入点与因特网接入服务提供商相连接,因此可以在接入点的广播范围内通过因特网接入服务提供商向终端系统提供因特网接入。接入点范围内的区域通常称为**热点**(hot spot)。热点和热点组的范围日益广泛,包括个人住宅、酒店和写字楼、小型企业、公园,甚至有些情况下是整个城市。目前手机行业也使用相似的技术,在手机行业中热点就是我们所知的服务区,当终端系统从一个服务区移动到另一个服务区时,通过调整产生服务区的“路由器”来提供连续的服务。

连接因特网接入服务提供商的其他常见技术使用电话线或者电缆/卫星系统。这些技术可以用来提供对单个终端系统的直接连接,或者连接客户的路由器从而实现连接多个终端系统。后

一种方法在个人住宅中的应用日益流行，利用现有的电缆或者电话线，通过连接因特网接入服务提供商的路由器/接入点形成本地热点。

现有的电缆和卫星链接与高速数据传输的兼容性天生要比与传统电话线的兼容性好，电话线最初是基于语音通信安装的。但是，人们已经制定了一些更明智的方案，拓展了这些语音链路，以适应数字数据的传输。这些连接使用了名为**调制解调器**（modem，modulator/demodulator的简称）的装置，该装置将数字数据转换为与使用的传输媒介兼容的格式。例如，DSL（Digital Subscriber Line，数字用户线路）把低于4千赫兹的频率范围用来满足传统的语音通信，而将更高的频率用来传输数字数据。此外，较为陈旧的方法是将数字数据转换为语音数据，然后使用和语音传输一样的方式进行传送。后一种方法称为**拨号连接**（dial-up access），事实上拨号连接用于临时连接，用户使用传统方式拨通因特网接入服务提供商的路由器，然后将其电话与要使用的终端系统连接。虽然拨号连接成本低并且使用广泛，但是数据传输速率较低，无法满足当今需要实时视频通信和传输大量数据的因特网应用。因此，越来越多的家庭和小型企业通过宽带技术连接他们的因特网接入服务提供商，这些宽带技术包括有线电视连接、专用电话数据线路、卫星天线，甚至光纤电缆。

4.2.2 因特网编址

如4.1节所介绍的，一个因特网必须与一个互联网范围的编址系统相连接，这个系统将赋予该系统中每个计算机唯一的标识地址。在因特网中，这些地址称为**IP地址**（IP address）。（IP是Internet Protocol的简称，指的是因特网协议，我们在4.4节中会更多地了解这个术语。）最初，每一个IP地址都是32位的位模式，但是为了提供更多的地址，人们现在正计划将其扩展到128位（参见4.4节关于IPv6的详述）。ICANN（Internet Corporation for Assigned Names and Numbers，因特网名称与数字地址分配机构）向因特网服务提供商提供了大量连续数字的IP地址，因特网名称与数字地址分配机构是一家非营利性的国际组织，致力于协调因特网的运营。然后，因特网服务提供商就可以将他们被授权范围内的地址块中的IP地址分配给其管辖范围内的计算机。因此，因特网上的计算机都被分配了唯一的IP地址。

IP地址通常是采用**点分十进制记数法**（dotted decimal notation）书写的，其中地址的每个字节用圆点分隔，每个字节用传统的十进制整数表示。例如，使用点分十进制记数法，位模式5.2将可以表示2字节位模式0000010100000010，其中包含字节00000101（5的表示），接下来是字节00000010（2的表示）；而位模式17.12.25将可以表示3字节的位模式，其中包含字节00010001（用二进制记数法表示17），然后是字节00001100（用二进制记数法表示12），最后是字节00011001（用二进制记数法表示25）。总之，当用点分十进制记法表示时，32位的IP地址可以表示为192.207.177.133。

用位模式表示的地址（即使采用点分十进制记数法压缩）难以帮助人们理解。基于这个原因，因特网拥有另外一种编址系统，利用助记名称来标识计算机。该编址系统基于**域**（domain）的概念，可以认为是如大学、俱乐部、公司或者政府机构等单个机构操作的因特网的“区域”。（此处区域一词加了引号，这是因为它可能不同于因特网的物理区域，稍后我们将看到这点。）每一个域都必须在因特网名称与数字地址分配机构进行注册，这一过程由称为**注册商**（registrar）的公司操作，注册商由因特网名称与数字地址分配机构指定。作为注册流程的一部分，助记**域名**会分配给域，域名在因特网中是唯一的。域名通常是注册域的机构的描述，从而提高它们对用户的实用性。

例如，Addison-Wesley出版公司的域名是aw.com。需要注意的是，点后面的后缀反映了域的分类，在这个域名中，后缀com就表明了它是一家商业机构（commercial）。这种后缀称为TLD

（Top-Level Domain，顶级域名）。顶级域名还有不少，例如，edu表示教育系统，gov表示政府机构，org表示非营利机构，museum表示博物馆，info表示无限制使用，net最初打算用于表示因特网服务提供商，但是现在使用的范围更广一些。除了这些一般的TLD外，也有用于表示特定国家的2字母TLD（称为**国家代码顶级域名**），例如，au表示澳大利亚，ca表示加拿大。

一旦一个域的助记名被注册，注册该域名的机构可以在域内自由扩展名称，为个体项获取助记标识符。例如，Addison-Wesley出版公司内的某台计算机可以标识为ssenterprise.aw.com。注意，域名是向左扩展的，并用句点分开。在一些情况下，使用称为**子域**（subdomain）的多个扩展在域内组织名称。这些子域通常代表域管辖内不同的网络。例如，如果Nowhere大学被分配的域名为nowhereu.edu，那么Nowhere大学的某台计算机的域名可以为r2d2.compsc. nowhereu.edu，其含义是顶级域名为edu、域名为nowhereu、子域名为compsc、名为r2d2的计算机。（我们需要注意的是，在助记地址中使用的点分表示法与位模式形式的地址中使用的点分十进制记数法没有关系。）

虽然助记地址对于人类来说比较方便，但是因特网中还是使用IP地址来传输消息。因此，如果某人想要给远程计算机发送消息并通过助记地址来标识目的地，那么使用的软件必须能够将地址转换成IP地址，然后再传输消息。这种转换可以通过**域名服务器**（name server）来完成，其实它是可以向客户端提供地址解析服务的目录。这些域名服务器都共同作为因特网范围内的目录系统，称为**DNS**（Domain Name System，域名系统）。使用域名系统进行解析的过程称为**域名系统查找**（DNS Lookup）。

因此，如果一台计算机可以通过助记域名连接，那么这个域名必须存在于域名系统内的域名服务器上。在注册了域名的实体拥有资源的情况下，它可以在域内建立并维护包含所有名称的域名服务器。事实上，这就是域名系统最初构建所用的模式。每一个注册域都代表了本地机构所运行因特网的物理区域，如公司、学校或者政府机构。实际上，这个机构就是一个因特网接入服务提供商，通过与因特网连接的内联网向其成员提供因特网接入。作为该系统的一部分，机构维护自己的域名服务器，为域中使用的所有名称提供解析服务。

目前，这种模式依然很常见。然而，许多个体或者小型机构想要建立展示在因特网上的域，但却不想承担必要的支持资源。例如，一家本地国际象棋俱乐部想要在因特网上展示为KingsandQueens.org，但是俱乐部不想提供资源建立自己的网络、维护网络与因特网的连接，也无意去实现自己的域名服务器。这种情况下，俱乐部可以和因特网服务提供商签订合同，使用因特网服务提供商已经建好的资源实现注册域名的展示。俱乐部一般可能会通过因特网服务提供商的帮助，注册由俱乐部选好的域名，并与因特网服务提供商签订合同将域名放入因特网服务提供商的域名服务器。这就意味着所有关于新域名的域名系统查找都会指向因特网服务提供商的域名服务器，从而获取正确的域名解析。通过这种方法，许多注册的域名就可以存在于一个因特网服务提供商的域名服务器中，每个域名只占用一台计算机的很小一部分。

4.2.3 因特网应用

在这一小节中，我们从3项传统的应用开始论述因特网的一些应用。但是，这些“传统”的应用不足以反映当今因特网的全貌。事实上，计算机和其他电子设备的区别已经变得含混不清。电话、电视、音响系统、防盗自动警铃、微波炉和摄像机都是潜在的“因特网设备”。此外，因特网的传统应用在新应用的大量扩展下相形见绌，新的应用包括即时消息、视频会议、因特网电话和因特网广播。毕竟，因特网只是一个传输数据的通信系统。随着技术进步，系统的传输速度不断提高，传输的数据内容就仅仅受到人们想象力的限制。因此，我们将介绍两种较新的

因特网应用（电话和广播），以展示与当今新兴因特网相关的问题，其中包括其他网络协议标准的需求，因特网与其他通信系统链接的需求，以及扩展因特网路由器功能的需求。

1. 电子邮件

因特网最常见的用途之一就是**电子邮件**（E-mail，是electronic mail的缩写），电子邮件系统可以在因特网用户之间传输信息。为了提供电子邮件服务，每个域的本地机构都要在其域内指定一台计算机，用作该域的**邮件服务器**（mail server）。通常，为了向域内的用户提供邮件服务，因特网接入服务提供商都会在其域内建立电子邮件服务器。当一名用户在其本地计算机上发送出电子邮件，邮件会首先传送到用户的邮件服务器上，然后邮件服务器会将邮件转发至目的地邮件服务器上，目的地邮件服务器会一直保存邮件，等待收件人联系邮件服务器并请求查看收到的邮件。

在邮件服务器之间传输邮件或者从作者的本地计算机向邮件服务器发送新消息，所使用的网络协议是SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）。简单邮件传输协议最初是为传输ASCII编码的文本信息设计的，因此又开发了MIME（Multipurpose Internet Mail Extensions，多用途因特网邮件扩展）等协议，将非ASCII编码的数据转换成简单邮件传输协议兼容的格式。

有两种常见的协议可以用于访问到达和存储在用户邮件服务器的电子邮件：POP3（Post Office Protocol Version 3，邮局协议版本3）和IMAP（Internet Mail Access Protocol，因特网邮件访问协议）。二者中，邮局协议版本3（POP3，读作pop-three）较为简单。使用POP3，用户可以向其本地计算机发送（下载）消息，在本地计算机中读取、在不同文件夹中存储，并可以随意编辑或者操作那些消息。这些操作都是通过使用本地机器的大容量存储器在用户的本地机器上完成的。IMAP（读作EYE-map）支持用户在与邮件服务器相同的计算机上存储和操作消息以及相关的资料。这样，必须在不同计算机上收取邮件的用户，就可以在邮件服务器上维护记录，然后通过任何远程计算机来访问。

了解邮件服务器的作用后，我们就很容易明白单个邮件地址的结构了。它有一个符号串（有时候称为账户名）表示某个人，然后是符号@（读做at），最后是助记串，表示接收该邮件的邮件服务器。（事实上，这个串通常只表明目标域，该域的邮件服务器最终是要通过DNS查找来表示的。）因此，Addison-Wesley的某个人的邮件地址很可能是shakespeare@aw.com的形式。换句话说，一条发送到这个地址的报文将发送到域名为aw.com的邮件服务器上，该报文一直保存在该邮件服务器中，以便符号串为shakespeare标识的人查看邮件。

2. 文件传输协议

传输文件（如文档、照片或者其他编码信息）的一种方法是将其作为附件附在电子邮件中。不过一种更有效的方法是利用FTP（File Transfer Protocol，文件传输协议），它是一种在因特网上传输文件的客户机/服务器协议。使用FTP传输文件，因特网中一台计算机的用户需要使用一个实现FTP的软件包，然后与另外一台计算机建立连接。（最初的计算机相当于客户机，它所连接的计算机相当于服务器，通常称为FTP服务器。）一旦建立了这个连接，文件就可以在两台计算机之间以任意方向传输了。

FTP已经成为因特网上提供受限数据访问的流行方式。例如，假设你打算允许一部分人检索某文件，但是禁止其他人访问，于是，你仅仅需要用FTP服务器设备将该文件存入一台计算机，然后通过口令限制对该文件的访问权限。然后，知道口令的人们就可以通过FTP访问该文件，而其他人的访问就会被拒绝。在因特网中使用这种方式的计算机通常被称为FTP站点，因为它成为因特网上可以通过FTP使用文件的地方。

FTP站点也用于提供不受限的文件访问。为了实现这个目的，FTP服务器要使用术语

anonymous（匿名）作为通用口令。这些地点经常被称为**匿名FTP**（anonymous FTP）站点，并因此提供不受限的文件访问权限。

虽然FTP客户机和服务器仍被广泛使用，但现在大部分用户发现他们的文件传输需要使用HTTP通过Web浏览器实现（将在下一节讨论）。

3. 远程登录与SSH

因特网的一个早期应用是，允许计算机用户在很远的地方访问计算机。**远程登录**（Telnet）就是为了这个目的建立的一个协议系统。使用远程登录，用户（运行远程登录客户端软件）可以与远程计算机的远程登录服务器取得联系，然后遵循特定操作系统的登录步骤获取对那台远程计算机的访问权。这样，通过远程登录，远程用户拥有与本地用户相同的对应用和工具的访问权限。

由于它设计于因特网发展初期，远程登录有许多缺点。其中一个较为严重的是，通过远程登录进行的通信是没有加密的。即使通信的主题不敏感，这个问题也很严重，因为用户的口令也是登录过程中通信的一部分。因此，使用远程登录就可能给窃听者获取口令的机会，然后他会滥用这一重要信息。SSH（Secure Shell，安全shell）是解决这个问题的一个远程登录候选方法，而且正在迅速代替远程登录。SSH的特征是，它给传输中的数据提供加密以及验证（4.5节），验证过程就是确定通信双方的身份。

4. VoIP

作为最近的因特网应用的一个例子，VoIP（Voice over Internet Protocol）利用因特网基础设施提供与传统电话系统类似的语音通信。最简单的形式下，VoIP由不同机器上的两个进程构成，这些机器通过P2P模型传输音频数据——这个方法本身没有明显的问题。但是，初始化和接受呼叫，把VoIP与传统电话系统对接，以及提供像紧急911通信这样的服务，诸如此类的问题都超出了传统的因特网应用范畴。此外，拥有国家传统电话公司的政府把VoIP看成是一种威胁，对它们征收很高的税，或彻底宣布它们为不合法。

现在有4种不同形式的VoIP系统在展开竞争。**VoIP软电话**（soft phone）通过P2P软件允许两个或多个PC共享一个电话，所需硬件只是一个扬声器和一个麦克风。VoIP软电话系统的中的一个例子就是Skype，它还为客户提供与传统电话通信系统的链接。Skype的一个缺点是它是一个专有系统，因而许多操作结构是不对外公开的，这意味着Skype用户必须要在没有第三方论证的基础上相信Skype软件的完整性。例如，为了接收呼叫，Skype用户必须把PC与因特网相连，并且Skype系统是可用的，这就意味着在PC机主毫无意识的情况下，PC的一些资源可能会被用来支持其他的Skype通信（这一功能引起了一些抵制）。

另一种形式的VoIP由**模拟电话适配器**（analog telephone adapter）组成。模拟电话适配器允许用户将其传统电话连接到由某个因特网接入服务提供商提供的电话服务上。这一选择经常与传统的因特网服务或数字电视服务捆绑。

第三种形式的VoIP是嵌入式VoIP电话。嵌入式VoIP电话把传统电话替换成了直接连接到TCP/IP网络上的等效的手持设备。嵌入式VoIP电话在大型组织中越来越常见，很多大型组织都在将其传统的内部铜线电话系统替换为基于以太网的VoIP，以期通过这种方式来降低成本并增强功能。

最后，下一代的智能手机将使用VoIP技术。也就是说，早期的几代无线电话仅使用特定公司的协议与该电话公司的网络通信。通过公司的网络和因特网间的网关，我们便可以连接到因特网，而信号会在那儿被转换为TCP/IP系统。然而，新兴的4G电话网络被设计成完全基于IP的网络，即4G电话本质上只是全球因特网上的另一主机。

无线电话的演进

在过去的10年时间里，移动电话技术已经从简单的专用便携设备发展成为复杂的多功能手持式计算机。第一代无线电话网络通过空气传输模拟语音信号，与传统的电话系统非常相似，只是没有穿墙而过的铜线。我们把这些早期的电话系统称为“1G”网络，即第一代网络。第二代网络使用数字信号对语音编码，能够更高效地使用无线电波，还能够传输其他种类的数字数据，如文本消息。第三代（3G）电话网络提供了更高的数据传输速率，支持手机视频通话和其他带宽密集型活动。4G网络目标包括更高的数据传输速率和一个使用IP协议完全进行分组交换的网络（以这个网络为基础的最新一代智能手机将能够具有当前仅在基于宽带的PC上可用的功能）。

5. 因特网广播

另一个最近的因特网应用是广播站节目的播送，这个过程称为网站广播，与原先的广播相对应，因为信号是通过因特网传送的，而不是无线电传送。更准确地说，因特网广播是**音频流**（streaming audio）的一个具体示例，它是指在实时基准上传送声音数据。

从表面上看，因特网广播似乎不需要特殊的考虑，人们可能猜想到一个站点仅需建立一台服务器，它把节目消息发送给请求它们的每个客户端。这种技术就是众所周知的**多点传播**（N-unicast）。（严格地说，单点传播是指一个发送者向一个接收者发送消息，而多点传播是指一个发送者从事多个单点传播。）多点传播方法已经投入实际使用，但这种方法具有明显的缺陷，它把相当大的负担放在站点服务器上和与服务器紧邻的因特网邻居上。实际上，多点传播强迫服务器按照实时基准把各条消息发送给每个客户端，而所有这些信息必须再由服务器的邻居转发。

多点传播的大多数候选方法都试图缓解这个问题。其中一种方法是使用过去的文件共享系统方法中的P2P模型。也就是说，一旦一个对等体接收到数据，它就开始把数据分发到那些正在等待的对等体，这意味着分发的大部分问题被从数据源转移到了对等体。

另外一种候选方法称为**多路广播**（multicast），它把分发问题转移给了因特网中的路由器。使用多路广播，服务器通过单个地址把消息传送给多个客户端，依赖因特网中的路由器来识别这个地址的含义，产生且转发消息的副本到合适的目的地。在多路广播中使用的这个地址称为组地址，它由特别的起始位模式来标识，其他的位用来标识广播站，这在多路广播术语中称为组。当一个客户端要从具体的站接收消息时（想要订购一个具体的组），它把这个愿望通知给最近的路由器。这个路由器本质上把这个请求向前传送到因特网，这样其他的路由器将会把所有带有这个组地址的信息向这个客户方向传送。简言之，当使用了多路广播，服务器只发送程序的一个副本，而不管有多少个客户在监听，把这些信息按需复制和把它们路由到合适的目的地则是路由器的职责。注意，依赖多路广播的应用需要因特网路由器具有超过它原来职责范围的功能。这一功能扩展过程正在进行之中。

我们看到因特网广播和VoIP一样，正在一边寻找立足之地，一边逐渐流行。未来到底会发生什么并不明确，但随着因特网基础设施功能的继续扩展，网站广播的应用肯定会随之发展。

现在，嵌入式设备和家用计算机已经能够通过因特网按需传送高清视频。各种电视机、DVD/蓝光播放机、游戏机现在都可以直接连接到TCP/IP网络，以便从众多免费或订阅服务器选择可视内容。

问题与练习

1. 第一层和第二层ISP的作用是什么？因特网接入服务提供商的作用是什么？

2. DNS是什么？
3. 用点分十进制记数法表示，3.6.9代表什么位模式？用点分十进制记数法表示位模式0001010100011100。
4. 计算机在因特网中的助记地址（例如r2d2.compsc.nowhereu.edu）结构在哪些方面与传统的邮寄地址相似？在IP地址中也会出现同样的结构吗？
5. 列出因特网上发现的3种服务器，并说出每种的用途。
6. 为什么人们认为SSH优于远程登录？
7. 因特网广播的P2P和多路广播方法与多点广播在广播方式上有何种不同？
8. 在4种VoIP中作出选择时，应以什么为标准？

4.3 万维网

本节将考虑一种因特网应用，多媒体信息就是通过它在因特网上传播的。它基于**超文本**（hypertext）的概念，超文本最初指的是包含指向其他文档的链接的文本文档，而这种链接称为**超链接**（hyperlink）。今天，超文本已经扩展到了包含图像、音频以及视频，而且由于其范围的扩大，它有时候也被称为**超媒体**（hypermedia）。

使用 GUI 时，超文本文档的读者只需要用鼠标点击超链接就可以获取与它相关联的内容。例如，假设语句“这个管弦乐队对 Maurice Ravel 的‘Bolero’的演奏精彩极了”出现在一个超文本文档中，名字 Maurice Ravel 与另外一个文档链接——这个新文档也许提供了该作曲者的介绍，则读者可以通过用鼠标点击名字 Maurice Ravel 查看相关信息。此外，如果安置了恰当的超链接，读者还可以通过用鼠标点击名字 Bolero 收听到该音乐会的录音。

通过这种方式，超文本文档的读者就可以查阅相关的文档，或者跟随思维顺序，一个文档一个文档地查看。各个文档的许多部分都与其他文档相链接，于是就形成了一个相关信息的相互“缠绕”的网状组织。当在计算机网络中实现时，存在于网状组织中的这些文档就可以存放于不同的计算机上，形成了网络范围的网状组织。在因特网上发展起来的网状组织已经遍布全球，被称为**万维网**（World Wide Web，也称 WWW、W3 或者 Web）。万维网上的超文本文档通常称为**网页**（Web page）。紧密相关的一组网页称为**网站**（Web site）。

万维网要源于 Tim Berners-Lee 所作出的努力，他意识到了链接文档的想法与互联网技术结合会产生巨大的潜力，并于 1990 年 12 月开发出了第一个实现万维网的软件。

4.3.1 万维网实现

允许用户访问因特网上超文本的软件包分为两类，即扮演客户角色的包和扮演服务器端角色的包。客户端软件包安装在用户的计算机上，负责获取用户请求的材料，并将这些材料条理清晰地展示给用户。客户端提供给客户一个界面，允许其在万维网上浏览。因此，客户端常被称为**浏览器**（browser）或者是万维网浏览器。服务器软件包（通常称为**万维网服务器**，Web server）放在含有待读取的超文本文档的计算机中。它的任务是根据客户端的请求提供对机器里文档的访问权。总之，用户通过他计算机里的浏览器访问超文本文档。充当客户端的浏览器通过向遍布因特网的万维网服务器提出请求服务来访问那些文档。超文本文档通常使用称为 HTTP（Hypertext Transfer Protocol，超文本传输协议）的协议在浏览器与万维网服务器之间传输。

为了在万维网上定位及检索文档，每个文档都被赋予了唯一的一个地址，称为 URL（Uniform Resource Locator，统一资源定位符）。每个 URL 都包含浏览器要连接到正确的服务器以及请求希望的文档所需要的信息。因此，为了浏览网页，人们要首先提供给浏览器所需要文档的 URL，

然后要求该浏览器检索和显示这个文档。

图 4-8 给出了一个典型的 URL，它包含以下 4 段：与控制文档的存取的服务器进行通信的协议，服务器所在的机器的助记地址，目录路径（供服务器找到存放该文档的目录），以及该文档的名字。简言之，图 4-8 中的 URL 告知浏览器：使用 HTTP 协议与称为 ssenterprise.aw.com 的计算机上的万维网服务器连接，并检索名为 Julius-Caesar.html 的文档，该文档存放在 authors 目录内的子目录 Shakespeare 中。

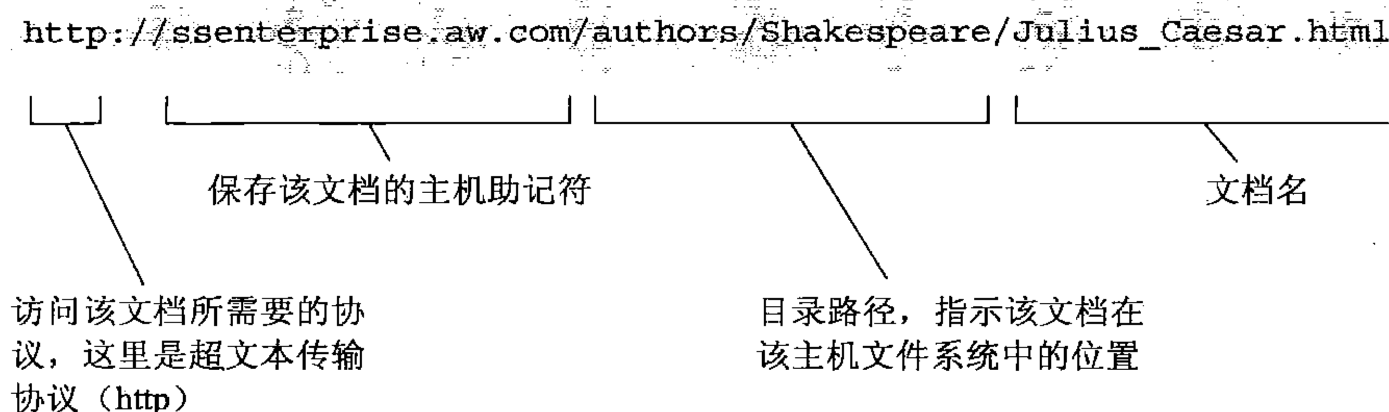


图4-8　典型的URL

有时候，URL 可能不会包含图 4-8 中所示的所有段。例如，如果服务器不需要根据目录路径去读那个文档，那么 URL 中就不会出现目录路径。此外，有时候一个 URL 只包含一个协议以及一台计算机的助记地址。在这些情况下，该计算机的万维网服务器将返回预定的一个文档，它通常称为主页，一般描述该网站上可用的信息。这种缩短了的 URL 使得联系各种机构的方法变得更简单。例如，http://www.google.com 这个 URL 表示谷歌公司的主页，它包含许多超链接，可以链接到与该公司相关的服务、产品和文档上去。

为了进一步简化网站定位，许多浏览器都假定：如果没有明确说明协议，就使用HTTP协议。当“URL”只包含www.google.com时，这些浏览器也能准确地检索到谷歌公司的主页。

万维网联盟

W3C（World Wide Web Consortium，万维网联盟）创建于1994年，宗旨是通过开发协议标准（称为W3C标准）来促进万维网的发展。W3C总部设在瑞士日内瓦欧洲粒子物理研究所（CERN）高能粒子物理实验室。CERN是原来的HTML标记语言和用于在因特网上传输HTML文档的HTTP协议的诞生地。今天的W3C制订了许多标准（包括用于XML和许多多媒体应用的标准），这些标准使得大量因特网产品彼此兼容。在网站http://www.w3c.org上，你可以了解到关于W3C的更多信息。

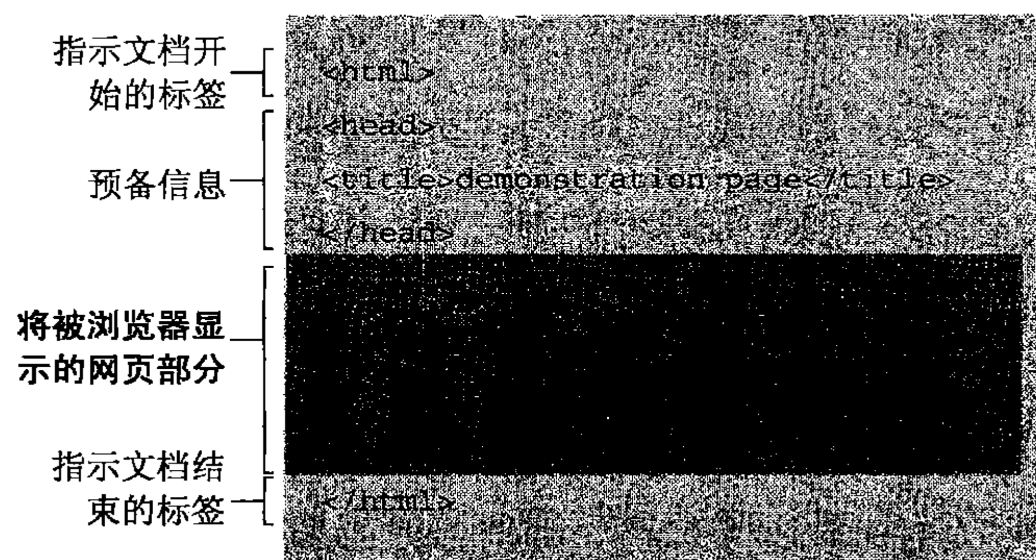
4.3.2　HTML

传统的超文本文档类似于文本文档，因为它的正文是使用诸如 ASCII 或者 Unicode 系统一个字符接一个字符地编码的。区别是，超文本文档还包含称为**标签**（tag）的专用符号，用于表示该文档应该如何呈现在显示器上，该文档还需要什么多媒体资源（如图像），以及该文档的哪些项链接到其他文档上。这个标签系统称为 HTML（Hypertext Markup Language，超文本标记语言）。

因此，按照 HTML 的要求，网页的作者描述了浏览器所需要的信息，使得浏览器能够将该页呈现在用户的显示器上，并找到当前网页所引用的任何相关文档。该过程类似于向简单输入

的文档加入排版命令（比如用红笔标注），以便排版人员知道文档最后应该以什么形式出现。对于超文本，HTML 标签就是这些红色记号，浏览器最终充当了排版人员的角色，读取 HTML 标签就会知道文本应以什么方式呈现在计算机显示器上。

图 4-9a 给出了一个非常简单的网页的 HTML 编码版本（称为源版本）。注意，标签是用符号"<"和">"括起来的。HTML 源文档由两段组成——首部（以标记<head>引导，以</head>结束）和主体（由<body>引导，以</body>结束）。网页首部和主体之间的区别类似于办公室备忘录的首部与主体之间的区别，两者都是：首部包含文档的预备性信息（例如备忘录的日期、主题等）；主体包含文档的实质内容，对于网页就是该页可能会呈现在计算机显示器上的内容。



(a) 用HTML编写的网页



(b) 显示在计算机屏幕上的网页

图4-9 一个简单的网页

图4-9a给出的网页的首部只包含了该文档的标题（用title标签括住）。该标题只是用于文档编制目的，并不会显示在计算机显示器上。要显示在显示器上的内容包含在文档的主体内。

图4-9a所示的文档主体的第一个条目是一个包含文本"My Web Page"的一级标题（在<h1>和</h1>标签之间）。一级标题意味着浏览器要将该文本明显地呈现在显示器上。主体的下一个条目是文本的一个段落（在<p>和</p>之间），包含文本"Click here for another page."。图4-9b给出的是浏览器显示在计算机显示器上的页面。

根据它现在的形式，图4-9所示的页面是没有实际意义的。用户单击here时，什么也不会发生，虽然网页上暗示了如此操作会使得浏览器打开另外一个网页。为了引发相关的动作，我们必须将单词here与另外一个文档建立链接。

假设单击here时，我们打算让浏览器呈现URL为http://crafty.com/demo.html的网页。首先，

我们要用标签<a>和在该网页源版本中将单词here括住，这对标签称为锚标签。在开锚标签里，我们要插入参数：

```
href = http://crafty.com/demo.html
```

（如图4-10a所示）它表示与该标签相联系的超文本引用（href）是在等号后面的URL（http://crafty.com/demo.html）。加上锚标签后，该网页就会如图4-10b所示呈现在计算机显示器上了。注意，它与图4-9b是一样的，只是单词here用彩色突出显示，表示它是到另外一个网页文档的链接。单击这类突出显示的术语就会使得浏览器检索并显示相关的网页文档。因此，网页文档通过锚标签在彼此之间建立起了链接。

```
<html>
<head>
<title>demonstration page</title>
</head>
<body>
<h1>My Web Page</h1>
<p>Click

   here

   for another page.</p>
</body>
</html>
```

含有参数的锚标签

闭锚标签

(a) 用HTML编写的网页



(b) 显示在计算机屏幕上的网页

图4-10　增强的简单网页

最后应该简要说明一下，图像是如何加入我们这个简单的网页的。为此，假设要插入的图像的JPEG编码是存储在Images.com的Images目录中的文件OurPic.jpg，且对那一位置的万维网服务器可用。这样，在HTML源文档的<body>标签后插入图像标签<img src="http://Images.com/Images/OurPic.jpg">，我们就可以命令浏览器在该网页的顶部显示该图像了。这告诉浏览器的是名为OurPic.jpg的图像应该在该文档的开头显示。（src是source的缩写，意思是等号后面的信息表示的是要显示的图像的来源）。浏览器发现这个标签时，就会传输给位于Images.com的HTTP服务器一条报文，请求OurPic.jpg图像，并且恰当地显示该图像。

如果我们将该图像标签移到文档的后面，就在</body>标签前面，那么该浏览器就会在该

网页的底部显示该图像。当然，在网页上定位图像还有许多复杂的技巧，但是现在我们不必关心这些问题。

4.3.3 XML

HTML 本质上是一个符号系统，文本文档及其外观都可以通过它编码成一个简单的文本文件。同理，我们也可以将非文本内容编码成文本文件，例如乐谱。乍一看，传统上表示音乐的五线谱，节拍以及音符都不符合文本文件规定的一个字符接着一个字符的格式。不过，我们可以通过另外一种符号系统来克服这个困难。精确地说，我们规定用`<staff clef = "treble">`表示五线谱的开始，用`</staff>`表示五线谱的结束，用`<time> 2/4 </time>`表示节拍号，用`<measure>`和`</measure>`分别表示小节的开始和结束，用`<notes> egth C </notes>`表示八分音符 C，等等。那么，文本：

```
<staff clef = "treble"> <key>C minor</key>
<time> 2/4 </time>
<measure> <rest> egth </rest> <notes> egth G,
egth G, egth G  </notes></measure>
<measure> <notes> hlf E </notes></measure>
</staff>
```

就可以用于编码图4-11所示的五线谱。使用这种符号，乐谱就可以作为文本文件编码、修改、存储和在因特网上传输。此外，还可以编写软件，将这类文件的内容以传统乐谱的形式表现出来，甚至可以用一个音乐合成器来演奏此音乐。
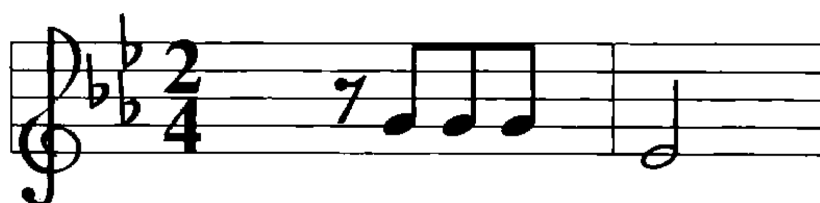


图4-11 贝多芬第五交响乐的前两小节

注意，我们的乐谱编码系统沿用了 HTML 使用的文体。对于标识组成部分的标签，我们选择用符号“<”和“>”作为定界符。我们选择用相同的标签名表示结构（如五线谱、一串音符，或者是一个节拍）的开始和结束，表示结束的那个标签有一条斜线（以`<measure>`开始的就以`</measure>`结束）。我们选择用诸如`clef= "treble"`的表达式来指示标签中特定的属性。这种文体还可以用于开发表示其他格式的系统，如数学表达式和图表。

XML（eXtensible Markup Language，可扩展标记语言）是一种标准化的文体（类似于上面乐谱的例子），用于设计将数据表示为文本文件的符号系统。事实上，XML是从比较老的称为SGML（Standard Generalized Markup Language，标准通用标记语言）的标准派生出的简化版。遵照XML标准，人们已经开发出了一批称为**标记语言**（markup language）的符号系统，可以表示数学、多媒体演示以及音乐。事实上，HTML就是一种基于XML标准开发的表示网页的标记语言。（实际上，HTML的原始版本在XML标准巩固之前就已经开发出来了，因此HTML的一些特征不严格遵守XML。正是这个原因，我们可能需要参考XHTML，它是严格遵守XML的HTML版本。）

关于如何设计标准以获得广泛应用，XML是个很好的范例。为了编码各种类型的文档，我们不应设计出单独的、无关联的各种标记语言，XML所用的方法是开发一种通用标记语言标准，通过这个标准，就可以为各种应用开发不同的标记语言了。这样开发出来的标记语言具有一致性，可以组合起来获得复杂应用的标记语言，例如包含乐谱片段和数学表达式的文本文档。

最后要说明的是，XML允许开发与HTML不同的新的标记语言，因为新的标记语言强调的是语义而不是词本身。例如，使用HTML可以标记菜谱里的配料，使得它们以列表形式出现，每一种配料单独占一行。不过，如果我们使用面向语义的标签，菜谱里的配料就可以标记为配料（也许使用标签<ingredient>和</ingredient>）而不仅仅是列表中的各个项。这个区别很细微但是很重要。语义方法使得**搜索引擎**（search engine，帮助用户搜索与特定主题相关的网页信息的网站）能够确定哪些菜谱包含或者不包含某些配料，这将是现在搜索引擎技术的一项重大改进，因为现在的技术只能够分离出含有或者不含有某些单词的菜谱。精确地说，如果使用语义标签，搜索引擎就可以找到不包含菠菜的卤汁宽面的菜谱，而只根据单词内容进行的类似搜索就会跳过以“这个卤汁宽面不包含菠菜”开始的菜谱。同样，如果根据语义而不是词本身使用因特网范围的标准标记文档，那么创建的是万维“语义”网，而不是现在使用的万维“语法”网。

4.3.4　客户端和服务器端的活动

现在我们来考虑一下，检索图4-10所示的简单网页，并将其呈现在浏览器所在的计算机的显示器上，都需要哪些步骤。首先，扮演客户端角色的浏览器要使用URL（也许是从使用该计算机的人那里获得）里的信息，与控制对该网页访问的万维网服务器建立连接，然后请求它传输该页的一个副本。服务器然后要将图4-10a上显示的文本文档发送给浏览器作为回应。接着，浏览器将解释该文档中的HTML标签，以确定如何显示该网页，并将文档呈现在计算机的显示器上。浏览器的用户就将看到如图4-10b所示的图像。如果用户接着用鼠标单击单词here，浏览器将使用相关联的锚标签中的URL链接到恰当的服务器，以获得并显示另一个网页。总之，整个过程就是浏览器按照用户的要求搜索及显示网页。

但是，如果我们想获得一个带有动画的网页，或者是一个允许客户填写订单并提交的网页呢？这些需求就需要浏览器或万维网服务器付出额外的行动。如果这些行动由客户机（如浏览器）完成则称为**客户端**（client-side）活动，如果由服务器（如万维网服务器）完成则称为**服务器端**（server-side）活动。

例如，假设旅行社想要客户能确认想去的目的地和旅行日期，它将呈现给客户一个定制的网页，只包含与该客户需求有关的信息。在这种情况下，旅行社的网站将首先呈现给客户一个包含可供选择的旅游目的地的网页。客户根据这个信息，指定感兴趣的目的地和旅行日期（客户端活动）。这些信息将传回给旅行社的服务器，服务器利用这些信息来构建合适的定制网页（服务器端活动），再将这个网页发送给客户的浏览器。

举一个搜索引擎服务的例子。在这种情况下，客户端的用户指定感兴趣的主题（客户端活动），然后这个主题被传送给搜索引擎，在那里会构建一个包含用户可能感兴趣文档的定制网页（服务器端活动），然后发回给客户端。还有一个例子就是**网站邮件**（Web mail）——一种日益流行的方法，通过它计算机用户能通过网页浏览器来访问他们的电子邮件。在这种情况下，万维网服务器是客户与客户邮件服务器间的中间层。从本质上讲，万维网服务器构建包含有来自邮件服务器信息的网页（服务器端活动），把这些网页传送给客户端，由客户端浏览器显示它们（客户端活动）。相反，浏览器支持用户创建消息（客户端活动），把这一信息传送给万维网服务器，万维网服务器再把这些消息转发给邮件服务器（服务器端活动）。

实现客户端和服务器端活动的系统有很多，可谓争奇斗艳，各有千秋。一种早期但现在仍然很流行的控制客户端活动的方法是，在网页的HTML源文档中包括用JavaScript语言（由网景公司开发）编写的程序。浏览器可以从中获取程序并根据需要执行。另外一种由Sun公司开发的方法是，首先将一个网页传输给浏览器，然后将称为小应用程序（applet，用Java语言编写）的

额外程序单元根据HTML源文档的要求传输给该浏览器。还有一种方法是使用由Macromedia公司[①]开发的Flash，可以实现大量多媒体客户端演示。

控制服务端活动的一个早期方法是，使用一组称为CGI（Common Gateway Interface，公共网关接口）的标准，客户通过它可以请求执行在服务器中存储的程序。这种方法的一个变体（由Sun公司开发）是允许客户在服务器端执行称为小服务程序（servlet）的程序单元。如果所请求的服务器端工作是创建定制的网页，如旅游代理的例子，那么就可以使用小服务程序方法的一种简化版本。这时，称为JSP（Java Server Page，Java服务器页面）的网页模板存放在万维网服务器里，并利用从客户端接收到的信息完成该网页。微软公司采用了类似的方法，构建定制网页的模板称为ASP（Active Server Page，活动服务器页面）。与这些专有系统相对，PHP是一种实现服务器端功能的开源系统。PHP最初代表个人主页（Personal Home Page），现在指的是PHP超文本处理程序（PHP Hypertext Processor）。

最后，我们应清醒地认识到，由于允许客户机和服务器在对方的计算机上执行程序，会引发一些安全性问题和道德问题。万维网服务器要固定地给客户传输要执行的程序，这个事实给服务器端带来了道德问题，并相应地给客户端带来了安全性问题。如果客户机盲目地执行万维网服务器发送来的任何程序，它就为服务器的恶意行动打开了大门。同理，客户机可以让程序在服务器端执行，因此也给客户端带来了道德问题，相应地给服务器端带来了安全性问题。如果服务器端盲目地执行客户机发送过来的任何程序，那么安全性就被破坏了，并因此产生潜在的危险。

问题与练习

1. 什么是URL？什么是浏览器？
2. 什么是标记语言？
3. HTML和XML的区别是什么？
4. 下列每个HTML标签的功能是什么？
   a. <html> b. <head> c. </p> d.
5. 客户端和服务器端分别指什么？

*4.4 因特网协议

本节研究报文是如何在因特网上传输的。因为传输过程需要系统中所有计算机合作，所以控制传输过程的软件需要驻留在因特网的每台计算机中。我们首先研究此类软件的总体结构。

4.4.1 因特网软件的分层方法

网络软件的首要任务是提供从一台机器到另一台机器传输报文所需的基础设施。在因特网上，报文传递活动是通过软件单元的层次结构完成的，这和你把一份礼物从美国的西海岸邮寄到东海岸的过程类似（见图 4-12）。第一步，把礼物打包并在包裹外面写上正确的邮寄地址；接着，把包裹拿到运输公司，例如邮局；运输公司把这个包裹和其他包裹一同放入一个大的集装箱，并送往与其签有服务合同的航空公司；航空公司将集装箱装入飞机并运往目的城市，也许沿途还要经过中转站；到达目的地，航空公司把集装箱从飞机上卸下，然后送往当地的运输公司；接着，运输公司把你的包裹从集装箱取出并送给收件人。

① 已被Adobe公司收购。

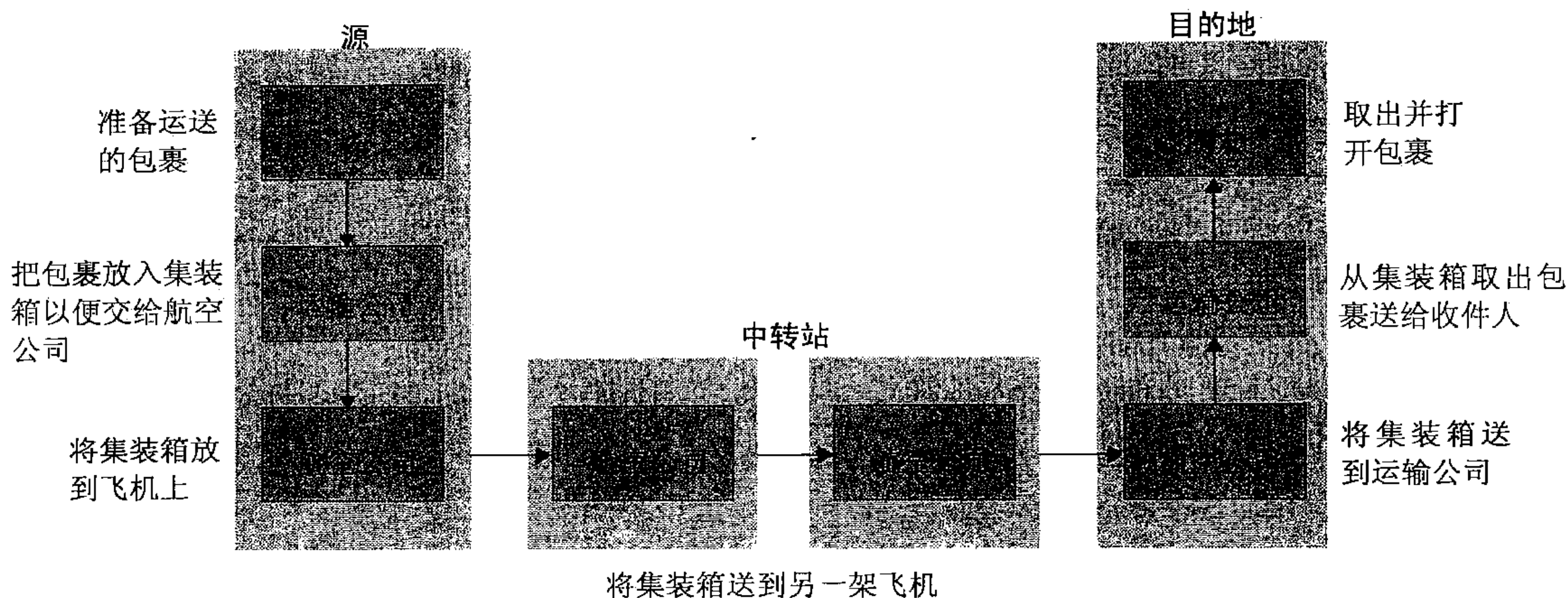


图4-12　包裹运输的例子

简而言之，礼物的运输过程需要3层组织：（1）用户层次（包括你和你的朋友）；（2）运输公司；（3）航空公司。每一层把下一层当做抽象工具来使用。（你不关心运输公司的工作细节，而运输公司也不需要关心航空公司的内部运作。）组织的每一层在源端和目的端都有代理，在目的端的代理完成在源端相应代理的相反工作。



图4-13　因特网软件层次

因特网上软件控制通信的过程和运输包裹的情况类似，但因特网上的软件有4层而不是3层，每层所涉及的是软件例程而不是人和企业。这4层就是众所周知的**应用层**（application layer）、**传输层**（transport layer）、**网络层**（network layer）、**链路层**（link layer）（图4-13）。通常，由应用层产生一个报文，当这个报文准备发送时，从应用层向下传递，经由传输层和网络层，最后传递到链路层进行传输。目的地的链路层接收这条报文，沿逆向分层结构向上传递，直到把它交给目的地的应用层。

下面通过跟踪一个报文通过网络系统的路径，从总体上研究一下报文的传输过程（参见图4-14）。首先从应用层开始。

应用层由那些使用因特网通信来完成任务的软件单元组成，如客户机和服务器软件。虽然名称类似，但是这一层不局限于3.2节介绍的软件分类中的应用软件，还包括一些实用软件包。例如，使用FTP协议传输文件的软件和利用SSH提供远程登录功能的软件已经很普遍，所以通常称它们为实用软件。

在因特网上，应用层使用传输层发送和接收报文的方式与我们通过运输公司邮寄和接收包裹非常相似。正像我们有责任提供一个和运输公司所要求的规范一致的地址一样，应用层负责向因特网基础设施提供兼容的地址。为了满足这个要求，应用层利用因特网上的域名服务器提供的服务把便于人类记忆的地址翻译成符合因特网规范的IP地址。

传输层的重要任务是从应用层接收报文并确保报文以正确的格式在因特网上传输。为了第二个目的，传输层将长报文分成小的片段作为独立单位在因特网上传输。因为在因特网内一个长报文会阻塞许多报文必经的节点，所以对长报文的分段是必需的。实际上，小段的报文在这些节点可以交叉通过，而一个长报文在经过这些节点时将迫使其他报文等待（很像在铁路交叉道口，许多小汽车等一列长火车通过的情况）。

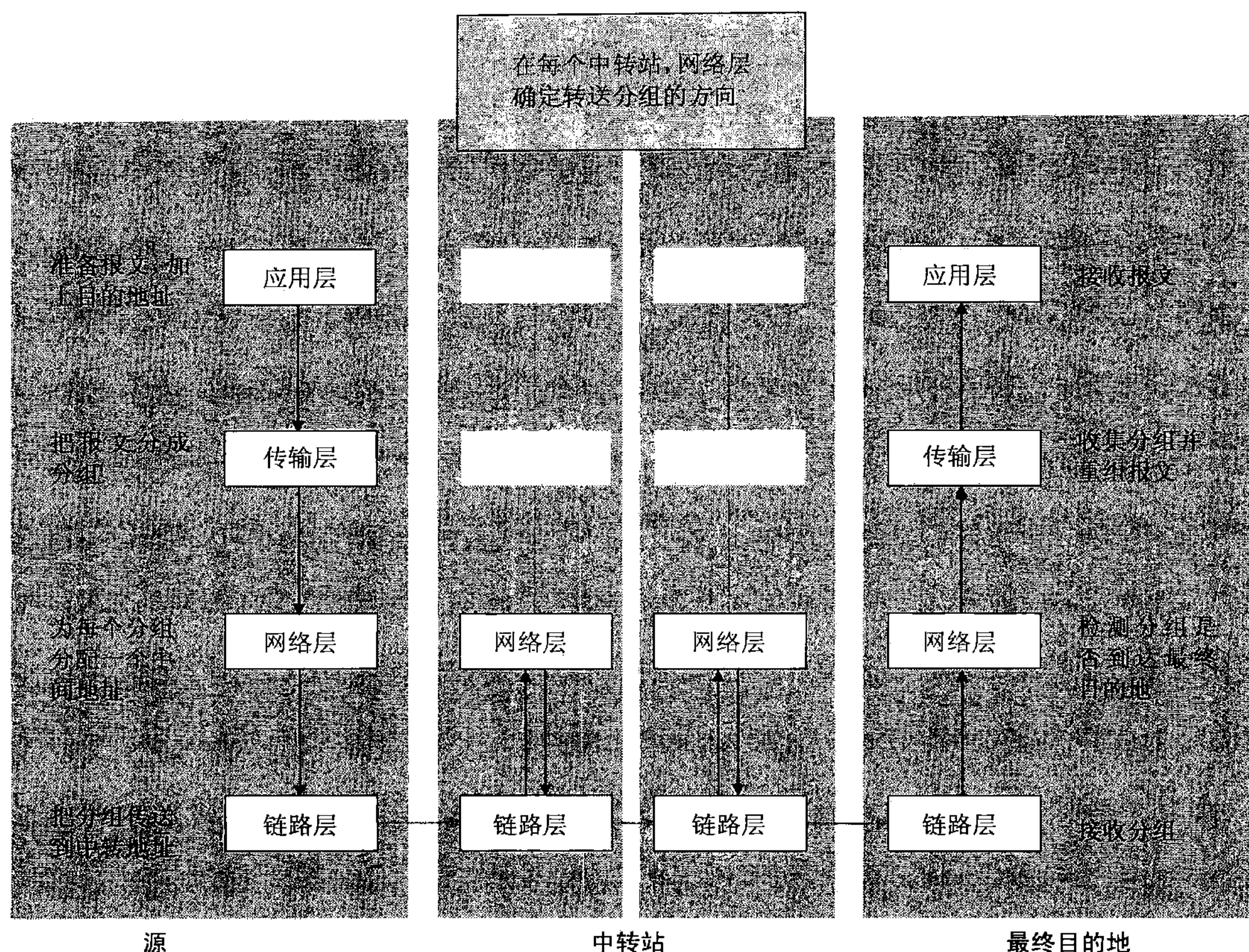


图4-14 因特网上报文的传输过程

传输层在生成的小片段上增加序列号，从而使这些片段在报文的目的地可以重新组合，然后将这些称为**分组**（packet）的片段交给网络层。从这一刻开始，这些分组被认为是彼此独立而无关的报文，直到它们到达最终目的地的传输层。这些属于同一个长报文的分组很有可能沿着不同的路径在因特网中传输。

在因特网的传输路径的每个步骤上决定分组的下一个发送方向，这是网络层的任务。事实上，网络层和其下层的链路层的组合构成了驻留在因特网路由器上的软件。网络层负责维护路由器的转发表，并使用此表决定分组的转发方向。路由器中的链路层负责接收和传输分组。

这样，当分组发源地的网络层接收来自传输层的分组时，它使用其转发表来决定分组应该被发送到哪里，并在那里开始分组的旅程。决定好合适的方向后，网络层把分组交给链路层，进行实际传输。

链路层具有传输分组的职责，因此，它必须要处理目的计算机所在的个体网络的特有的通信细节。例如，如果网络是以太网，链路层将使用CSMA/CD协议；如果网络是WiFi网，链路层将使用CSMA/CA协议。

当分组被传输后，它被处在连接另一端的链路层接收到。在那里链路层把分组向上交给网络层，由网络层把分组的最终目的地和网络层的转发表进行比对，决定分组下一步的方向。当作出这个决定后，网络层把分组返回给链路层，分组沿着它的路径被转发。使用这样的方式，分组从一台机器跳到另一台机器，最终到达它的目的地。

在这个旅程中，只涉及中转站的链路层和网络层（再次参见图4-14），因此正如前面提及的，只有这两个层存在于路由器上。此外，为了尽量减少在每个中转站上的延迟时间，路由器中的

网络层转发角色是紧密地与链路层集成在一起的。这样，现代路由器转发一个分组所需的时间是用微秒来衡量的。

在分组的最终目的地，由网络层来确认分组的旅程已经完成。此时，网络层把分组交给它的传输层，而不再转发它。传输层从网络层接收这个分组时，它提取组成报文的基本片段，并按照报文源端传输层所提供的片段序列号重组原始的报文。一旦报文重组了，传输层就把它交给应用层的适当单元——这样便完成了报文的传输过程。

确定应用层内哪个单元来接收到来的报文是传输层的一个重要任务，这个任务由为每个单元分配唯一的**端口号**（port number，它与第2章讨论的I/O端口无关）来控制，传输层在报文开始传输旅程之前要把适当的端口号附加到报文地址上。然后，一旦目的地的传输层收到了报文，它只需将报文交给指定端口号上的应用层软件。

因特网用户很少需要关心端口号，因为对于普通的应用有通用的端口号。例如，如果请求万维网浏览器检索URL为http://www.zoo.org/animals/frog.html的文档，那么浏览器认为要通过80端口和www.zoo.org的HTTP服务器联系。同样，当传输文件时，FTP客户机认为应当通过20和21端口与FTP服务器通信。

概括地说，因特网上的通信涉及4层软件的相互作用。应用层以应用的观点处理报文；传输层把报文转换成适合因特网的段，并负责将接收到的报文重组好后交给适当的应用程序；网络层处理段通过因特网的方向；链路层处理段从一个机器到另一个机器的实际传输。令人惊讶的是，虽然有这么多的工作，因特网的响应时间却是以毫秒记的，所以许多事务是瞬间完成的。

4.4.2　TCP/IP 协议簇

由于需要开放式网络，所以需要颁布一些标准，通过这些标准，不同制造商生产的设备和软件可以和其他厂商的产品一起正常运行，其中已产生的一个标准是由国际标准化组织制定的OSI（Open System Interconnection，开放系统互连）参考模型。与我们刚刚描述的4层结构不同，OSI标准基于7层结构。因为带有国际组织的权威性，所以它经常被引用，但是它已经很难取代4层结构的观点，这主要是因为4层结构在OSI模型制定之前已经成为因特网的事实标准。

TCP/IP协议簇是因特网所使用的协议的集合，这个协议集用来实现因特网的4层通信层次结构。实际上，TCP（Transmission Control Protocol，传输控制协议）和IP（Internet Protocol，网际协议）只是这个庞大集合中两个协议的名字——因此把这个协议集合称为TCP/IP协议簇容易产生误解。更确切地说，TCP定义了传输层的一个版本，这里说版本是因为TCP/IP协议簇不只提供一种传输层实现方式，例如还可用UDP（User Datagram Protocol，用户数据报协议）定义其他版本的传输层。传输层具有多种版本的情况和运输包裹的情况类似，你可以选择不同的运输公司，每家公司提供相同的基本服务，但又各有所长。因此，根据对特定的服务质量的要求，应用层的软件单元可以选择通过传输层的TCP版本还是通过UDP版本来传输数据（图4-15）。

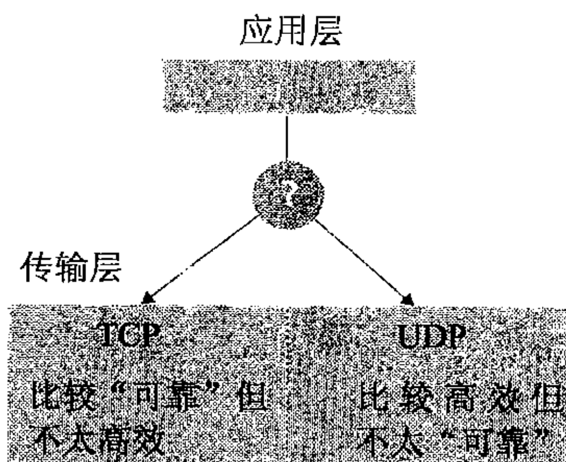


图4-15　TCP和UDP之间的选择

TCP和UDP之间有一些差别。第一个区别是，基于TCP协议的传输层发送应用层所请求的报文前，先要向目的地的传输层发送一个自己的报文，告诉目的地传输层有报文要发送，然后，它要等待目的地确认这个报文后才开始发送应用层报文。因此我们说基于TCP的传输层在发送报文前要建立一个连接。而基于UDP的传输层在发送报文前不需要建立这样的连接，它仅仅按照所给的地址发送报文，然后就忘记这个报文，尽管它知道目的地的计算机甚至可能是不运转的。由于这个原因，UDP被称为无连接协议。

TCP和UDP之间的第二个差别是，源和目的地的TCP传输层通过确认和分组重发的方式共同确保一个报文的所有片段都被成功地传输到目的地。因此，TCP被称为可靠的协议，而UDP不提供这种重发服务，被称为不可靠的协议。

TCP和UDP之间还有另外一个区别，那就是TCP提供了**流量控制**（flow control）和**拥塞控制**（congestion control），前者是指报文源点的TCP传输层能降低它发送数据段的速率，防止目的地的对应方应接不暇，后者是指报文源点的TCP传输层能调整它的发送速率，缓和它与报文目的地间的拥塞。

所有这些并不意味着UDP是一个不好的协议，要知道基于UDP的传输层比基于TCP的更简单。因此，如果一个应用有能力处理UDP的潜在影响，那么基于UDP的传输层会是更好的选择。例如，UDP的高效使得它成为DNS查找和VoIP选择的协议。但是，因为电子邮件在时间上不太敏感，所以邮件服务器使用TCP传输电子邮件。

IP是实现赋予网络层任务的因特网标准。我们注意到这个任务包括**转发**（forwarding，这涉及通过因特网传递分组）和**路由**（routing，这涉及更新层的转发表，以反映出条件的改变）。例如，一个路由器可能会出现故障（意味着这个方向上的信息不能再向前传输）或因特网的一个区域可能变得拥堵（意味着信息的传输应该绕过这个区域）。当相邻网络层交换路由信息时，与路由有关的IP标准大多数处理相邻网络层间的通信协议。

与转发有关的一个有趣的特性是，在信息的源头IP网络层每一次准备分组时，它都把一个称为**跳数**（hop count）的值（也叫生存的时间，time to live）加到分组上。这个值限制了分组在穿越因特网时被向前转发的次数。IP网络层每次向前转发一个分组，都要把这个分组的跳数减1。通过这个信息，网络层能保护因特网，以免分组在系统内无休止地循环。虽然因特网的规模每天都在增长，但初始的64跳数仍然足以让分组在当今ISP的路由器迷宫中找到它自己的出路。

多年以来，称为IPv4（版本4）的IP版本一直被用于在因特网内实现网络层，然而因特网迅速超出了IPv4所规定的32位互联网编址系统。为了解决这一问题并实现其他改进（如多路广播），人们建立了一个称为IPv6的新IP版本，它使用128位的互联网地址系统。目前，IPv4正在向IPv6过渡（4.2节中介绍因特网地址的部分间接提到过这个过渡），预期2025年前因特网中的32位地址将绝迹。

问题与练习

1. 因特网软件层次结构的哪些层不需要用在路由器上？
2. 基于TCP协议的传输层和基于UDP协议的传输层之间有哪些区别？
3. 传输层如何确定应用层的哪个单元应当接收到来的消息？
4. 怎样阻止因特网上的计算机记录所有途经它的报文的副本？

4.5 安全性

当一台计算机连接到网络上时，它会遭受到未授权用户的访问和恶意破坏。本节讨论和这

些内容有关的话题。

4.5.1 入侵的形式

通过网络连接侵袭计算机系统以及其所存储的资源有很多种方法，大多数方法会用到**恶意软件**（malware）。这类软件可以在某台计算机内部扩散和运行，也可以侵袭远距离的计算机。病毒、蠕虫、特洛伊木马和间谍软件都是以入侵的方式在计算机中扩散和运行的恶意软件，这些名称反映了软件的主要特征。

计算机应急响应小组

1988年11月，发布到因特网上的一个蠕虫病毒造成了网络服务的严重瘫痪。随后，美国国防高级研究计划署（DARPA）成立了CERT（Computer Emergency Response Team，计算机应急响应小组），并设立在卡内基-梅隆大学内的CERT协调中心。CERT是因特网安全的“监督者”，其职责是调查安全问题、发布安全警告和发起运动提高公众的因特网安全意识。CERT协调中心的网站为http://www.cert.org，上面有其活动的公告。

病毒（virus）是这样一种软件，首先它通过将自身嵌入到计算机已有的程序来感染计算机。接着，当“宿主”程序运行时，病毒也运行。许多病毒运行时仅仅把自身扩散到计算机中的其他程序，但是有些病毒会执行破坏性的动作，例如，使操作系统的部分性能下降，删除海量存储器的大数据块，或者毁坏数据和其他程序。

蠕虫（worm）是有自主能力的程序，它可以通过网络传播，占据计算机的存储空间并通过复制扩散到其他计算机。和病毒的情况一样，蠕虫可以只是用来复制自身或实施较严重的破坏行为。蠕虫破坏的一个典型后果是，蠕虫副本的激增会使合法程序的性能下降，最终整个网络或互联网因为负载过重而瘫痪。

特洛伊木马（Trojan horse）是一种伪装成合法程序（比如游戏或有用的实用程序包）进入计算机系统的软件，它们被受害者自愿引入。然而，一旦特洛伊木马程序进入计算机，它就会实施额外的破坏活动。这些活动有时是立即发生，而有时特洛伊木马可能暂时处于休眠状态，直到被一个特殊的事件激活，如一个预定日期的到来。特洛伊木马常常以有诱惑力的电子邮件附件的形式出现，打开这类附件（确切地说是接收邮件者请求浏览附件）时，它的破坏活动就开始了，所以决不要打开来源不明的电子邮件附件。

恶意软件的另一种形式是**间谍软件**（spyware），有时称为**嗅探**（sniffing）软件，这类软件收集它所驻留计算机的活动信息，并把这些信息报告给攻击的发起者。有的公司使用间谍软件来建立目标客户档案，这么做值得质疑的是它是否违背道德。对于其他情况，使用间谍软件的目的就是用来破坏，比如通过记录计算机键盘的打字序列，来寻找密码或信用卡卡号。

间谍软件通过秘密嗅探方式获取信息，与此相反，**电子黑饵**（phishing）技术是简单直接地索要信息。由于电子黑饵的诈骗过程是向网络中撒大量的“线”，来等待某些人上钩，所以电子黑饵术语是钓鱼的双关语[1]。电子黑饵通常用电子邮件来实施，这与老式电话诈骗没有差别。诈骗犯以金融机构、政府机关或者执法机构的名义发送邮件信息，向可能的受害者索要假装用于合法目的的信息，而实际上，这些信息被诈骗犯恶意使用。

① phishing与fishing的发音相同。——译者注

与遭受病毒和间谍软件等内部传染不同，计算机还能够被网络系统中其他计算机所运行的软件攻击。DoS（Denial of Service，拒绝服务）攻击就是其中的一个例子，这个程序使得计算机接收超负荷的要求回复的信息。拒绝服务攻击已经向因特网的大型商业万维网服务器发起攻击，从而破坏公司的业务，有时甚至曾导致一些公司的商业活动中断。

拒绝服务攻击要求在短暂的时间内产生大量的要求回复的信息，为了达到这个目的，攻击者通常在大量未设防的计算机内植入攻击软件，只要给一个信号，攻击软件就会产生要求回复的信息。接着，一旦信号发出，所有这些计算机就用要求回复的信息将目标计算机淹没。因而，拒绝服务攻击的实质是把未设防的计算机作为帮凶来使用。这就是为什么应该劝诫所有的PC用户，在不使用因特网时断开网络。据估计，PC一旦连接到因特网，20分钟内就至少有一个入侵者会尝试利用它。因此，未设防的PC严重威胁因特网的安全性。

另一个和无用信息有关的问题是散布无用垃圾邮件，我们称**垃圾邮件**为spam。和拒绝服务攻击不同的是，垃圾邮件的数量不足以压垮计算机系统。但是，垃圾邮件的作用是压垮接收垃圾邮件的人。正如我们所看到的，这一问题已经被复杂化了，垃圾邮件被大量用作电子黑饵和特洛伊木马传播病毒及其他恶意软件。

4.5.2 防护和对策

毫无疑问，“防患于未然”道出了控制网络连接上恶意破坏情况的真理。一个主要的防护技术是过滤穿过网络某一重要节点的通信流，通常使用称为**防火墙**（firewall）的软件。例如，防火墙可能安装在组织内联网的网关处来过滤进出区域的信息。这种防火墙设计的目的是，阻止向某些特定目的地址发送信息以及阻止接受已知的有问题的来源所发送的信息。后一种功能可以用来终止拒绝服务攻击，因为它阻止了来自具有攻击性计算机的通信量。安装在网关处的防火墙发挥的另一个作用是，阻止所有源地址为区域内地址的信息进入网关，因为这样的信息表明有外人假装区域内成员。把自己伪装成其他成员的行为称为**欺骗**（spoofing）。

防火墙不但能保护整个网络或域，更能用于保护个人计算机。例如，如果一台计算机不用作万维网服务器、域名服务器或邮件服务器，那么安装在这台计算机上的防火墙应当阻止所有用于这些应用的通信。实际上，入侵者获得计算机入口的一个途径就是通过一个已经不存在的服务器所留下的“漏洞”来建立联系。尤其是，利用间谍软件获取信息的方法就是在感染的计算机上建立一个秘密的服务器，通过这个服务器恶意客户端可以获取间谍软件的嗅探结果。正确安装防火墙可以阻止这类恶意客户端的报文。

还有些防火墙的变种是为一些特殊目的设计的，**垃圾邮件过滤器**（spam filter）就是其中一例，设计这种防火墙是为了阻止一些垃圾邮件。许多垃圾邮件过滤器在区分正常邮件和垃圾邮件时采用了相当复杂的技术。一些过滤器通过一个训练式的过程来学会这种区分判断，先由用户确定哪些属于垃圾邮件，过滤器获得了足够多的例子后可以自行作出判断。这些过滤器展现了将各种不同的学科领域（如概率论、人工智能等）联合起来，可以推动其他领域的发展。

另一种防护工具也具有过滤功能，它就是代理服务器。**代理服务器**（proxy server）是一个软件单元，它作为客户机和服务器之间的媒介，目标是保护客户机屏蔽来自服务器的不利行为。如果不用代理服务器，客户机就直接与服务器通信，这就意味着服务器有机会获得客户机一定量的信息。由于同一个组织的内联网内的许多客户机都与远程的服务器通信，长此以往，该服务器就能收集关于内联网内部结构的大量信息，而这些信息在以后可被用于怀有恶意的活动。为了防范这一点，组织可以建立一个代理服务器，用于特定种类的服务（如FTP、HTTP和远程

登录服务等），每次内联网内的客户机试图连接某个类型的服务器时，实际上连接的是代理服务器。于是代理服务器扮演了客户机的角色与实际的服务器相连。此后代理服务器就一直扮演着实际的客户机与实际的服务器之间的媒介，来回地中转报文。这种设置的第一个好处就在于，实际的服务器没办法知道代理服务器不是真的客户机，事实上它永远不会意识到实际客户机的存在。这样一来，实际的服务器就没有办法了解内联网的内部特性。第二个好处在于代理服务器能够过滤服务器发往客户机的所有报文。例如，FTP代理服务器能够检查所有的进入文件，看是否感染了当前已知的病毒，然后阻止所有感染了病毒的文件进入。

另一种用于防止网络环境中的问题的工具是**审计软件**。它类似于我们在操作系统安全问题中所讨论的审计软件（见3.5节）。通过网络审计软件，系统管理员能够察觉到管辖范围内不同位置突然激增的报文流量，监控系统防火墙的活动状态，并且可以对个人计算机的请求模式进行分析，用以探测网内的非正常行为。审计软件是管理员依靠的主要工具，用于及早发现问题。

另一种防御通过网络连接进行入侵行为的方法就是采用**防病毒软件**（antivirus software）。这种软件用来探测和删除被已知病毒或其他方式感染的文件。（实际上，防病毒软件代表了一大类软件产品，每一种软件产品都设计成探测和删除某一特定类型的感染文件。例如，许多产品专门研究如何防控病毒，另外一些产品则专门研究怎样防范间谍软件。）这些软件包的用户需要理解，正如生物系统一样，新的计算机病毒感染会不断地出现，因而需要不断地更新疫苗。因此，防病毒软件必须要从软件提供商那里定期下载更新。然而，即使是这样也不能保证计算机的绝对安全。毕竟，新病毒在被发现和其相关的疫苗产生之前一定是先感染了一些计算机的。因此，明智的计算机用户应该做到以下几点：决不打开一个不熟悉来源的电子邮件中的附件，也不要没有确认软件的可靠性就下载该软件，不要轻易响应弹出广告，在PC没有必要连接在因特网上时，不要将其连接上因特网。

4.5.3 加密

有时网络中恶意行为的目的是使系统瘫痪（如拒绝服务攻击），但有时其最终目的是获取信息的访问权。保护信息的传统方法是通过口令来控制对信息的访问。然而，当数据通过网络和互联网传输时，报文会被一些未知的实体进行中转，因而口令安全性就会受到威胁，因此没有多大的价值。在这样的情况下，可以使用加密技术，使得即使这些数据落入不怀好意的人的手中，编码后的信息依然能保持其机密性。在今天，许多传统的因特网应用已经进行了改变，并与加密技术相结合，因而也就产生了这些应用的所谓“安全版本”。这样的例子包括**FTPS**（即FTP的安全版本）以及SSH（我们在4.2节中介绍过，它是远程登录服务的安全替代产品）。

还有另一个例子，即HTTP的安全版本，称为**HTTPS**，它用于大多数金融机构中，并为客户账号提供安全的因特网访问。HTTPS的核心是称为**SSL**（Secure Sockets Layer，安全套接字层）的协议系统，它最初是由网景公司开发的，用来为万维网中的客户机和服务器提供安全的通信链路。大多数浏览器通过在计算机屏幕上显示一个很小的锁的图标来表明SSL已启用。（有的会用图标的出现与否来表示是否正在使用SSL，有的通过显示锁是闭合还是打开的来表示是否用了SSL。）

在加密领域里一个更令人着迷的话题就是**公钥加密**（public-key encryption）。在这个系统中，即使知道是如何对报文进行加密的，也无法知道如何对报文进行解密。这个特性看起来好像有些违反直觉，毕竟直觉告诉我们，如果一个人知道怎样对报文进行加密，那么他就应该能够反

向地对报文进行解密。但是，当使用公钥加密技术时，这个直觉就是错误的。

PGP

因特网上最流行的公钥加密系统恐怕是基于RSA算法的，该算法的名字来源于其创立者Ron Rivest、Adi Shamir和Len Adleman，对此我们将在第12章末详细讨论。RSA技术（和其他技术）已经用在了PGP公司（PGP代表Pretty Good Privacy）开发的一系列软件包中。这些软件包与PC上使用的大多数电子邮件软件都兼容，并且个人用户可以免费获得，用于非商业用途，详见http://www.pgp.com。通过PGP软件，个人用户可以生成公钥和私钥，并用公钥对报文进行加密，用私钥对报文进行解密。

公钥加密系统涉及两个称为**密钥**（key）的值的使用。一个密钥称为**公钥**（public key），用来对报文进行加密；另一个密钥称为**私钥**（private key），用来对报文进行解密。使用这个系统时，首先将公钥分发给那些需要向某个目的地发送报文的一方，而私钥则在这个目的地端机密地保存。于是，报文发起方可以用公钥对报文进行加密，然后将该报文送往目的地，即使在这期间被其他知道公钥的中间人截获，也仍能保证它的内容是安全的。事实上，唯一能对报文进行解密的是在报文的目的地持有私钥的那一方。这样一来，如果Bob创建了一个公钥加密系统，并把公钥给Alice和Carol这两个人，那么Alice和Carol这两个人都能对发往Bob的报文进行加密，但是他们不能够窥探对方的通信。确实是，如果Carol截获了来自Alice的报文，即使她知道Alice是怎样进行加密的，也不能对该报文进行解密（见图4-16）。

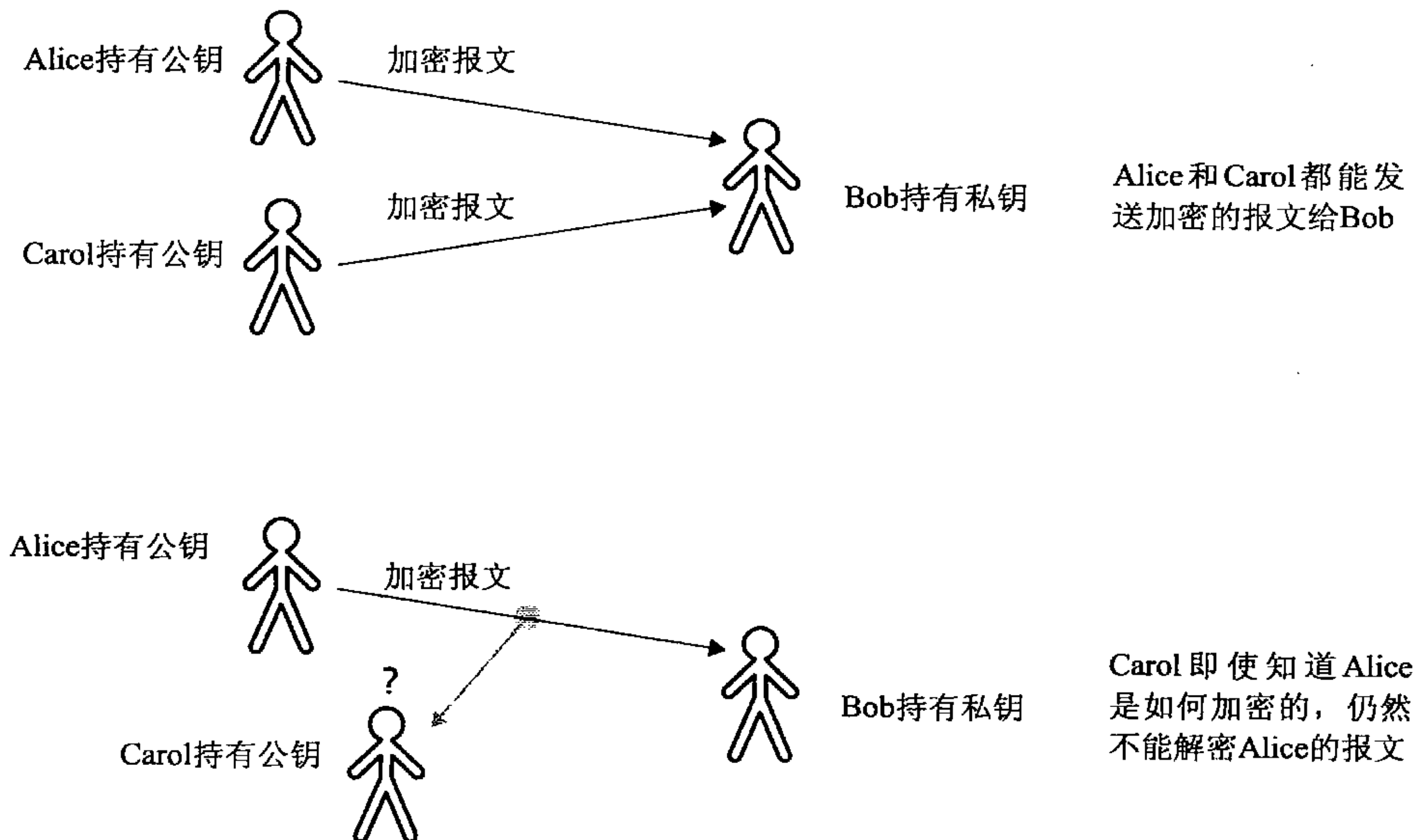


图4-16 公钥加密

当然，公钥系统中存在一些小问题。一个问题就是，要保证所用的公钥事实上对目的地的那一方而言是一个正确的密钥。举例来说，如果你正在和银行通信，你想确定这样一个事实，即你用来加密的公钥是针对银行而言的，而不是针对一个冒名顶替者。如果一个冒名顶替者让自己以银行的身份出现（行使欺骗），并把它的公钥给你，那么你就会对报文进行加密，并发送给“银行”，这个信息对这位冒名顶替者（而不是银行）非常有意义。因此，将公钥关联到正确的另一方的任务很重要。

解决这个问题的一个办法就是建立一些可信任的因特网站点，称为**认证机构**（certificate authority），其任务是维护相关方的准确列表以及他们的公钥。于是，这些起着服务器作用的认证机构为他们的客户提供了可靠的公钥信息，以称为证书的软件包的形式来提供。**证书**（certificate）是一个软件包，它包含有关方的名称和该方的公钥。现在在因特网上有许多商业认证机构，但为了更有效地保持对其通信安全性的控制，由组织来维护他们自己的认证机构也是件比较常见的事。

最后，我们应该在解决**鉴别**（authentication）问题方面对公钥加密系统进行一下说明，鉴别就是要确定报文的作者实际上确实是他们声称的那一方。这里关键的问题就在于，在有些公钥加密系统中，加密密钥和解密密钥的作用可以转换。也就是说，原文可以由私钥来加密，并且因为只有一方可以访问这个密钥，所以这样加密的任何原文都必须是从那一方产生的。在这种方式下，私钥的持有者就能产生一个位模式，称为**数字签名**（digital signature），只有那一方才知道应怎么生成。通过对报文附加这样的签名，发送者就能对报文做可以信任的标记。数字签名可以和报文本身的加密形式一样简单。所有的发送方必须做的事情就是对要发送的报文用自己的私钥（这个密钥通常用作解密）进行加密。当接受方收到报文时，就利用发送方的公钥对这个签名进行解密。这样得出的报文就能保证其可信性，这是因为只有私钥的持有方才能产生该报文的加密形式。

4.5.4　网络安全的法律途径

另一种增强计算机网络系统安全性的方法就是应用法律补救措施。然而，这种方法有两个障碍。第一个障碍在于认定一个行为不合法，并不意味着会排除该行为，而只是提供了一个法律依靠而已。第二个障碍在于网络的国际特性意味着要获得追索权通常是很困难的。在一个国家中不合法，但在另一个国家却可能是合法的。最终，通过法律途径来增强网络安全性是一个国际性的问题，所以必须由国际法律机构来处理。一个可能的机构将是位于海牙的国际法庭。

尽管这么说，我们必须承认，虽然法律措施并不完美，但还是有很大影响力的。所以对我们而言，在网络领域里，研究用来解决冲突的一些法律步骤还是有必要的。为此目的，可以用美国联邦法案来作为例子进行说明。另外，还可以从其他一些政体，如欧盟等，找到类似的例子。

首先讨论恶意软件的繁殖问题。在美国，这个问题是由《计算机欺诈和滥用法》（Computer Fraud and Abuse Act）提出的，该法案于1984年首次通过，其后已经做了几次修改。通过这个法案，涉及蠕虫和病毒制造的大多数案例都已经被起诉。简而言之，这个法案需要证据证明，被告有意引起一段程序或数据的传播，而这个程序或数据有意地进行了破坏。

《计算机欺诈和滥用法》还针对涉及信息盗窃的案例。具体来说，这个法案规定，通过非授权的方式访问计算机并获取任何有价值的信息的行为均不合法。法院已经对“任何有价值的”赋予了广泛的解释，所以《计算机欺诈和滥用法》已经不仅仅适用于信息盗窃的情况。例如，法院规定，仅仅是使用了计算机就可以算作是“任何有价值的”。

在法律界，隐私权是另一个也许是最富有争议的网络问题。这样的问题包括雇主是否有权监视员工的通信，以及因特网服务提供商在多大程度上有权访问其客户正在交流的信息，这些问题已经得到了相当多的关注。在美国，这些问题有许多已经在1986年的ECPA（Electronic Communication Privacy Act，《电子通信隐私法》）中提到，这个法案起初是为控制搭线监听设立的。虽然法案很长，但是仍能从几段短的摘录中捕捉到它的意图。比如，它声明：

> 除了本章中特别提到的，任何有意截获、力图截取或者唆使他人截取或力图截取任何有线、口头或电子通信……的人应按照子条款(4)受到惩罚，或者按照子条款(5)受到起诉。

还有

> ……任何向公众提供电子通信服务的个人或实体，不得在服务时有意将任何通信的内容……泄露给除了这些通信的收件人或意想的接收人，或者这些收件人或意想的接收人的代理人之外的任何人。

简而言之，ECPA确认了个人秘密通信的权利，因特网服务提供商泄露有关其客户的通信信息是非法的，并且非授权用户偷听他人的通信也是非法的。但是，ECPA还是留下了一些有争论的地方。例如，关于雇主监视雇员的通信的权利问题变成了一个授权问题，在这个问题上，当雇员用雇主的设备实施通信时，法院倾向于给予雇主这项权力。

此外，这个法案在某些条件限制下，会给某些政府部门监控电子通信的权利。这些规定已经引发了很多争论。例如，在2000年，FBI披露了它拥有一个Carnivore系统，该系统能报告一个因特网服务提供商的所有订户的通信信息，而不仅仅是法庭认可的目标。在2001年，为了回应针对世界贸易中心的恐怖袭击，国会通过了富有争议的USA PATRIOT（Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism，美国爱国者）法案，该法案修改了政府部门所受的限制。

提供这种监控权利除了引起了法律和道德上的争论外，还引起了与我们的研究更相关的一些技术问题。一个问题是，为提供这些能力，必须构建和编制通信系统，使其可以监控通信。建立这样的能力是CALEA（Communication Assistance for Law Enforcement Act，《通信辅助法》执行法案）的目标。它要求电信运营商修改它们的设备以适应法律强制监听，而这个需求一直比较复杂，实现起来也非常昂贵。

另一个富有争议的问题涉及政府监控通信的权利与公众使用加密的权利之间的冲突。如果正被监控的报文加密得很好，那么窃听通信对于法律强制机构来说就没有多大价值。美国、加拿大和欧洲各国政府正在考虑需要注册加密密钥的系统，但是这样的需求受到了企业界的反对。毕竟，由于商业间谍的存在，很容易理解要求注册加密密钥会使得许多遵守法律的公司和个人感到不舒服。注册系统的安全性有多高？

最后，作为识别因特网环境的法律问题的范畴的一种工具，我们引用1999年的《反网络域名抢注消费者保护法》（Anticybersquatting Consumer Protection Act），设计这个法案是为了防止冒名顶替者建立一个看上去相似的域名（这个阴谋就称为域名抢注）来欺骗大家。这个法案禁止使用与其他商标或“民法商标”一样的或相似得容易引起混淆的域名。一个作用是，尽管该法案没有将域名的投机买卖（就是注册一个可能有需求的域名，以后再将该域名的所有权卖出的一个过程）视为不合法，但是它限制了对常用域名的投机买卖。因此，域名的投机买卖者可能能够合法地注册一个常用域名，如GreatUsedCars.com，但是如果Big Al公司已经在从事二手车业务，那么他就不可能注册域名BigAlUsedCars.com。这种区别经常会在与《反网络域名抢注消费者保护法》相关的法律诉讼案中成为争论的主题。

问题与练习

1. 什么是电子黑饵？如何保护计算机来抵御电子黑饵？
2. 能放在域网关的防火墙和放在域内单个主机上的防火墙类型之间有什么差别？

3. 理论上说，术语数据指信息的表示，而信息指基本的含义。口令用来保护数据还是信息？加密保护的是数据还是信息？
4. 和传统的加密技术相比，公钥加密技术的优势是什么？
5. 和防备网络安全问题的法律尝试相关的问题有哪些？

复习题

（带*的题目涉及选读小节的内容。）

1. 什么是协议？说出本章介绍的3个协议，并描述每个协议的目标。
2. 描述客户机/服务器模型。
3. 描述对等模型。
4. 说出3种分布式计算机系统。
5. 开放式网络和封闭式网络之间的区别是什么？
6. 为什么CSMA/CD协议不能应用于无线网络？
7. 描述在使用CSMA/CD协议的网络内，一台机器发送报文所要遵循的步骤。
8. 什么是隐藏终端问题？描述解决它的技术。
9. 集线器和中继器怎样区分？
10. 路由器和中继器、网桥及交换机这类设备怎样区分？
11. 网络和因特网的区别是什么？
12. 说出网络中用来控制报文发送权的两个协议。
13. 使用32位因特网地址原先被认为是提供了足够大的扩展空间，但这推测被证实并不准确。IPv6使用128位地址，这将被证明是足够的吗？证明你的答案。（例如，你可以把可能的地址数目与世界的人口进行比较。）
14. 用点分十进制记数法为下列位模式编码。
    a. 0000010100010010001000100011
    b. 1000000000100000
    c. 0011000000011000
15. 下列点分十进制记数法表示的位模式分别是什么？
    a. 0.0
    b. 26.19.1
    c. 8.12.20.13
16. 假设因特网上一台终端系统的地址是134.48.4.122，那么这个32位地址如何用十六进制记数法表示？
17. 什么是DNS查找？
18. 如果一台计算机的助记因特网地址是batman.batcave.metropolis.gov，推测一下该机器所在域的结构是什么样的？
19. 解释电子邮件地址kermit@animals.com的组成。
20. 在VoIP语境下，模拟电话适配器和嵌入式电话的区别是什么？
21. 邮件服务器的功能是什么？
22. 多点传播与多路广播的区别是什么？
23. 给出下列名词的定义。
    a. 域名服务器
    b. 因特网接入服务提供商
    c. 网关
    d. 终端系统
24. 给下列名词的定义。
    a. 超文本
    b. HTML
    c. 浏览器
25. 因特网的许多“外行用户”经常混用因特网和万维网这两个术语。这两个术语的正确含义是指什么？
26. 在浏览一个简单的网页文档时，让浏览器显示该文档的源版本，然后说出该文档的基本结构。特别是，说出该文档的首部和主体，并列出你在首部和主体中发现的一些语句。
27. 列出5种HTML标签，并说出它们的含义。
28. 修改下面的HTML文档，使单词Rover链接到URL为http://animals.org/pets/dogs.html的文档。

```
<html>
<head>
<title>Example</title>
</head>
<body>
<h1>My Pet Dog</h1>
<p>My dog's name is Rover.</P>
</body>
</html>
```

29. 画一个草图来说明下面的HTML文档在计算机屏幕上的显示信息。

```
<html>
<head>
<title>Example</title>
</head>
<body>
<h1>My Pet Dog</h1>
<img src = "Rover . jpg">
</body>
</html>
```

30. 使用本章介绍的非正规XML风格来设计一个标记语言，把简单的代数表达式表示为文本文件。
31. 使用本章介绍的非正规XML风格设计一组标签，文字处理器可能用到这些标签标记潜在的文本。例如，一个文字处理器如何指示出什么文本应该是粗体、斜体、有下划线等？
32. 用本章介绍的非正规XML风格来设计一组标签，使得可以根据文本项在打印页上的出现方式给电影评论做标记。然后再设计一组标签，可以用于根据文本中这些项的含义标记电影评论。
33. 用本章介绍的非正规XML风格来设计一组标签，可以用于根据文本项在打印页上的出现方式给运动比赛项目的文章做标记。然后再设计一组标签，可以用于根据文本中这些项的含义标记这些文章。
34. 说出下面URL的组成，并描述各项的含义。
    http://lifeforms.com/animals/moviestars/kermit.html
35. 说出下列缩写URL的组成。
    a. http://www.farmtools.org/windmills.html
    b. http://castles.org/
    c. www.coolstuff.com
36. 如果要浏览器在下列两个URL"找文件"，浏览器的动作有什么不同？
    http://stargazer.universe.org
    https://stargazer.universe.org
37. 给出万维网上两个客户端活动的例子和两个服务器端活动的例子。

*38. 什么是OSI参考模型？

*39. 在一个基于总线型拓扑结构的网络里，总线对于要传送报文的机器是必须竞争的不可共享资源。在这种环境里死锁（参见选读的3.4节）是如何控制的？

*40. 列出因特网软件层次结构的4层，并说明各层所完成的任务。

*41. 为什么传输层把长报文划分为小分组？

*42. 当某应用程序要求传输层使用TCP来传输报文时，为了满足应用层的要求，传输层需要附加什么样的报文？

*43. 在实现传输层时，什么情况下TCP优于UDP？什么情况下UDP优于TCP？

*44. 说UDP是无连接协议的含义是什么？

*45. 在TCP/IP协议层次结构里，为了用下列方法过滤进来的通信流，防火墙应该设置在哪一层？
    a. 报文内容
    b. 源地址
    c. 应用类型

46. 假定你想建立一个可以过滤掉包含某些术语和短语的电子邮件报文的防火墙。这个防火墙应该放在域的网关上，还是放在域的邮件服务器上？说明理由。
47. 什么是代理服务器以及使用代理服务器的好处？
48. 总结公钥加密的原理。
49. 一台空闲但未设防的PC是如何威胁因特网的？
50. 对于整个因特网界限制用法律来解决因特网存在的问题，你如何看待？

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 计算机网络使得在家办公的观念流行起来。这种变化有哪些利弊？它对自然资源的消费有什么影响？它会使家庭巩固吗？它会减少"办公室政治"吗？在家里办公的人和在现场办公的人会有同样的职务晋升机会吗？社会联系会被削弱吗？减少和同行之间的个人接触会有正面的还是负面的影响？

2. 网上购物正在变成“亲身”购物的一个替代行为。这种购物习惯的变化对于社会有什么影响？对于大型购物中心的影响呢？对于你通常只逛不买的，比如书店和服装店之类的小店呢？以尽可能最低价格购买，好到什么程度，坏到什么程度？你是否有这样的道义上的责任，多花一点钱购买一个商品来支持本地的商业？比较本地商店里的商品，然后通过因特网以较低的价格订购，这合乎道德吗？这种行为的长期影响会是什么？
3. 政府对其公民访问因特网（或其他国际性网络）的控制应当限制在什么程度内？对于涉及国家安全的问题呢？可能发生哪些安全问题？
4. 电子公告牌允许网络用户发布消息（常以匿名方式）和阅读其他用户发布的消息。这个公告牌的管理人员应该对内容负责吗？电话公司应该对电话的通话内容负责吗？食品杂货店的管理者要对店内的社团公告牌内容负责吗？
5. 因特网的使用应当被监视吗？应当被管制吗？如果需要，应该由谁来管理，管理到什么程度？
6. 你花费多少时间来使用因特网？那些时间花得值吗？上网改变你的社会活动了吗？你认为通过因特网与人交谈比面对面与人交谈更容易吗？
7. 当你为个人计算机买了一个软件包的时候，开发商通常要你向开发商注册，以便你可以得到未来升级的通知。这种注册过程越来越多地通过因特网处理，经常要你提供诸如姓名、地址以及如何知道产品等信息，然后开发商的软件自动把这些数据传输给开发人员。如果开发商设计的注册软件在注册过程还把额外的信息发送给开发人员，那么会发生什么道德问题吗？比如，注册软件可能扫描你的系统内容，报告找到的其他软件包。
8. 访问一个网站时，这个站点有在计算机内记录数据（称为cookie）的能力，从而表明你曾经访问过该站点。然后这些cookie可被用来识别回访的访问者并记录他们以前的活动，以便网站可以更高效地应对访问者对网站的未来访问活动。计算机上的cookie还可提供你访问网站的记录。网站应该有在计算机内记录cookie的功能吗？未经你的同意，是否应允许网站在你的计算机里记录cookie？cookie可能的好处是什么？使用cookie可能会引发什么问题？
9. 如果政府机构要求公司注册加密密钥，公司还是安全的吗？
10. 一般来说，出于礼貌我们不会为了安排周末外出之类的个人或社团的事情而给在工作场所的朋友打电话。类似地，大多数人也不愿意打电话到客户的家里介绍新产品。按照类似的习俗，我们把婚礼请柬寄到客人的住所，而把商务会议的通知邮寄到出席者的工作地址。把给朋友的私人电子邮件通过他工作的地方的邮件服务器发送合适吗？
11. 假定一个PC的所有者让这台PC接入因特网，但最终这台电脑被其他人用来进行拒绝服务攻击。这个PC的所有者该负多大的责任？你的回答和他是否安装了正确的防火墙有关吗？
12. 一个生产糖果和玩具的公司在他们的公司网站上提供游戏，在推荐公司产品的同时让孩子们娱乐，这种做法道德吗？如果游戏是用来收集小孩信息的，那又如何？娱乐、广告和利用之间的界限是什么？

课外阅读

Antoniou, G. and F. van Harmelem. *A Semantic Web Primer*. Cambridge, MA: MIT Press, 2004.

Bishop, M. *Introduction to Computer Security*. Boston, MA: Addison-Wesley, 2005.

Comer, D. E. *Computer Networks and Internets*, 5th ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

Comer, D. E. *Internetworking with TCP/IP*, vol. 1, 5th ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Goldfarb, C. F. and P. Prescod. *The XML Handbook*, 5th ed. Upper Saddle River, NJ: Prentice-Hall, 2004.

Halsal, F. *Computer Networking and the Internet*. Boston, MA: Addison-Wesley, 2005.

Harrington, J. L. *Network Security: A Practical Approach*. San Francisco: Morgan Kaufmann, 2005.

Kurose, J. F. and K. W. Ross. *Computer Networking: A Top Down Approach Featuring the Internet*, 4th ed. Boston, MA: Addison-Wesley, 2008.

Peterson, L. L. and B. S. Davie. *Computer Networks: A Systems Approach*, 3rd ed. San Francisco: Morgan Kaufmann, 2003.

Rosenoer, J. *CyberLaw, The Law of the Internet*. New York: Springer, 1997.

Spinello, R. A. and H. T. Tavani. *Readings in CyberEthics*. Sudbury, MA: Jones and Bartlett, 2001.

Stallings, W. *Cryptography and Network Security*, 4th ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Stevens, W. R. *TCP/IP Illustrated*, vol. 1. Boston, MA: Addison-Wesley, 1994.

第5章 算法

在“绪论”一章中我们得知，计算机科学的核心主题是对算法的研究。现在是我们关注这个核心主题的时候了。我们的目标是探究足够的基本素材来真正地理解和认识计算科学。

本章内容

我们已经知道，在计算机能够执行一个任务之前，必须给出一个算法精确地告诉计算机做什么。因此，算法的研究是计算机科学的基石。在本章中，我们将介绍算法研究的许多基本概念，包括算法的发现和表示以及算法的主要控制结构——迭代和递归等问题。在讲解这些问题的同时，我们还会介绍几个有关查找和排序的著名算法。下面首先介绍算法的概念。

5.1 算法的概念

在“绪论”一章中，我们把算法非正式地定义为描述如何完成任务的步骤集。在本节中，我们将进一步讨论算法的基本概念。

5.1.1 概览

在前面的学习中，我们已经遇到了多个算法。我们发现了用来进行数制转换的算法、检测和纠正数据错误的算法、压缩和解压缩数据文件的算法、在多任务环境中控制多道程序设计的算法以及很多其他算法。此外，我们已经看到，CPU 所遵循的机器周期只不过是下面这个简单算法。

只要未发出停机指令就执行以下步骤：
a. 取一条指令；
b. 解码该指令；
c. 执行该指令。

就像图 0-1 中的魔术的算法所展示的那样，算法并不局限于技术活动，实际上它甚至可以用来描述剥豌豆壳这样的普通活动。

获得一篮子未剥壳的豌豆和一只空碗。只要篮中还有未剥壳的豌豆就执行下面的步骤：
a. 从篮子里拿出一个豌豆；
b. 剥开豌豆的豆荚；

c. 把剥落的豆放到碗里面；
d. 扔掉空豆荚。

实际上，许多研究人员相信，人脑中的每一个活动，包括幻想、创造和决策，实际上都是算法执行的结果——我们将在学习人工智能（第 11 章）时介绍。

但是，在继续深入研究之前，让我们先考虑一下算法的正式定义。

5.1.2 算法的正式定义

非正式、不严格地定义的概念在日常生活中是可接受的并且是很常见的，但是科学必须基于严谨定义的术语之上。现在，我们就来考察一下图 5-1 中算法的正式定义。

算法是定义一个可终止过程的一组有序的、无歧义的、可执行的步骤的集合。

图5-1 算法的定义

注意，该定义要求一个算法中的步骤集合是有序的。这意味着，一个算法中的各个步骤必须有一个就执行顺序而言非常明确的结构。这并不意味着这些步骤必须从第 1 步到第 2 步，再到下一步，这样顺序执行。有些算法，称为并行算法，包含的步骤序列不只一个，每一个序列都被设计成由多处理器机器中的不同处理器执行。在这种情况下，整个算法并不只包含一个遵照第 1 步、第 2 步这样的顺序的执行流，其结构是一种多执行流结构，这些执行流在整个任务的不同部分被不同的处理器执行时不断分支和再接合（我们会在第 6 章中再次讨论这个概念）。其他例子包括第 1 章中所讲述的触发器电路执行的算法，这个电路中每一个门电路都完成整个算法的一步。这里，这些步骤是按照因果关系排列的，每个门电路的结果都是通过电路传播的。

接下来，我们考虑算法必须由可执行的步骤组成这一要求。为了满足这个条件，我们考察下面这条指令：

给出一个所有正整数的列表。

由于正整数有无穷多个，完成这条指令是不可能的。因此，任何包括这条指令的指令集都不能称作一个算法。计算机科学家使用有效的（effective）这个术语来表示可执行的概念。也就是说，说算法中的一个步骤是有效的就意味着它是可执行的。

图 5-1 中的算法定义的另外一个要求是算法中的步骤必须是无歧义的。这意味着在算法的执行过程中，正在被处理的信息必须足以唯一地、完整地确定每一步所需要的动作。换句话说，算法中的每一步的执行都不需要创造性的技能，只要求遵照指令执行。（在第 12 章我们将学习的算法称为非确定性算法，但那个算法不受这里的限制，属于另外一个重要的研究论题。）

图 5-1 给出的定义还要求，算法定义的是一个可终止的过程，也就是说，一个算法的执行必须能够最终结束。这个要求源自理论计算机科学，其目标是要回答诸如“算法和机器的最终限制是什么？”之类的问题。其中，计算机科学试图寻找下面两种问题的区别：哪些问题的答案的获得在算法系统能力范围之内，哪些问题的答案的获得超出了算法系统能力范围。从这个意义上讲，它在以一个答案告终的过程与一个只能向前执行而不能得到结果的过程之间存在着一条分割线。

可是，还是有一些不可终止的过程是非常有意义的，包括监视病人的生命特征和维持飞行器的飞行高度等。有些人可能辩称这些问题仅仅是算法的重复，这其中的每一个算法都会在到达结束状态之后自动重复执行。另外一些反对这一论点的人可能认为，这种说法只不过是一种对于正式定义限制的过度坚持。不管是哪种情况，结果都是算法这个名词通常使用在对步骤集合

的实用的或者非形式的引用，并不一定必须定义一个可终止的过程。例如，当1除以3的时候，长除法“算法”并不定义一个可终止的过程。从技术上讲，这些例子都表示对该术语的误用。

5.1.3　算法的抽象本质

强调算法与其表示的区别是非常重要的，这就好像一个故事和一本书的差别。一个故事本质上是抽象的，或者说是概念上的，而一本书是一个故事的物理表示。如果一本书被翻译成其他语言或者以另外一种样式出版，仅仅是这个故事的表示形式改变了，而故事本身并没有变化。

同样，算法是抽象的，与它的表示是有差别的。一个算法可以用多种方式来表示。比如，在华氏温度和摄氏温度之间进行转换的算法可以用下面的代数公式表示：

$$F=\left(\frac{9}{5}\right)\mathrm{C}+32$$

但也可以用下面的指令表示：

将摄氏温度数值乘以$\frac{9}{5}$，然后在乘积上加32。

甚至可以用电路的形式予以表示。无论哪种情况，基本的算法是一致的，只不过是表示方式不同罢了。

算法和它的表示的区别体现了我们在传达一个算法的时候存在的问题。一个常见的例子是，一个算法必须描述到什么样的细致程度。对于气象学家来说，指令“将摄氏度读数转换为相应的华氏度数”就足够了，但是，对于一个外行来说（需要更加详细的描述）这个指令可能是模糊的、有歧义的。然而，问题并不在于算法，而是算法并没有很好地按照外行所要求的细致程度进行表示。在5.2节中，我们会学习原语的概念，并看到原语是如何被用于消除算法表示中的这种歧义性问题的。

最后，在算法和它的表示这个问题上，我们应该明确另外两个概念的区别——程序和进程。程序是一个算法的表示。（这里，我们没有在正式意义上使用术语算法，因为许多程序是不可终止的“算法”的表示。）实际上，计算机科学家通常用程序这个词表示那些设计成计算机应用程序的算法的表示。在第3章中，我们把进程定义为执行程序的活动。然而，请注意执行一个程序就是执行由该程序所表示的算法，所以一个进程可以等价地定义为执行一个算法的活动。我们可以得到这种结论：程序、算法和进程既是不同的却又有关联的。程序是算法的表示，而进程又是执行算法的活动。

问题与练习

1. 简述进程、算法和程序之间的区别。
2. 给出一些你所熟悉的算法的例子。它们是精确意义上的算法吗？
3. 对于在0.1节中给出的算法的非正式定义，存在哪些有歧义的（含糊的）地方？
4. 从什么意义上说，由下列指令列表所描述的步骤不能构成算法？

   第1步：从你的口袋里取出一枚硬币并且把它放到桌子上。

   第2步：返回第1步。

5.2　算法的表示

在本节中，我们考虑与算法表示有关的问题。我们的目标是引入原语和伪代码的基本概念，

并且建立一种为我们所用的算法表示系统。

5.2.1 原语

一个算法的表示需要使用某种形式的语言。对于人类，这可能是一种传统的自然语言（英语、西班牙语、俄语、日语），也可能是一种图形语言，如图 5-2 所示，在这个图中，我们给出了用一块正方形纸片叠出一只鸟的算法。然而，这种自然的沟通方式常常会引起误解，有些时候是因为算法描述中使用的术语可能拥有多种含义。例如，句子“Visiting grandchildren can be nerve-racking.”可能表示孙子来访时会惹出事情，也可能表示去看孙子是一件很费周折的事情。所需的详细程度引发的误解也可能导致问题。很少有读者能够按照图 5-2 给出的步骤成功地叠出一只小鸟来，但是一个专门学习折纸的学生可能很轻松地就将其完成。简言之，当用来描述算法的语言并没有被准确定义或者并没有给予足够详细的信息的时候，交流就会产生问题。

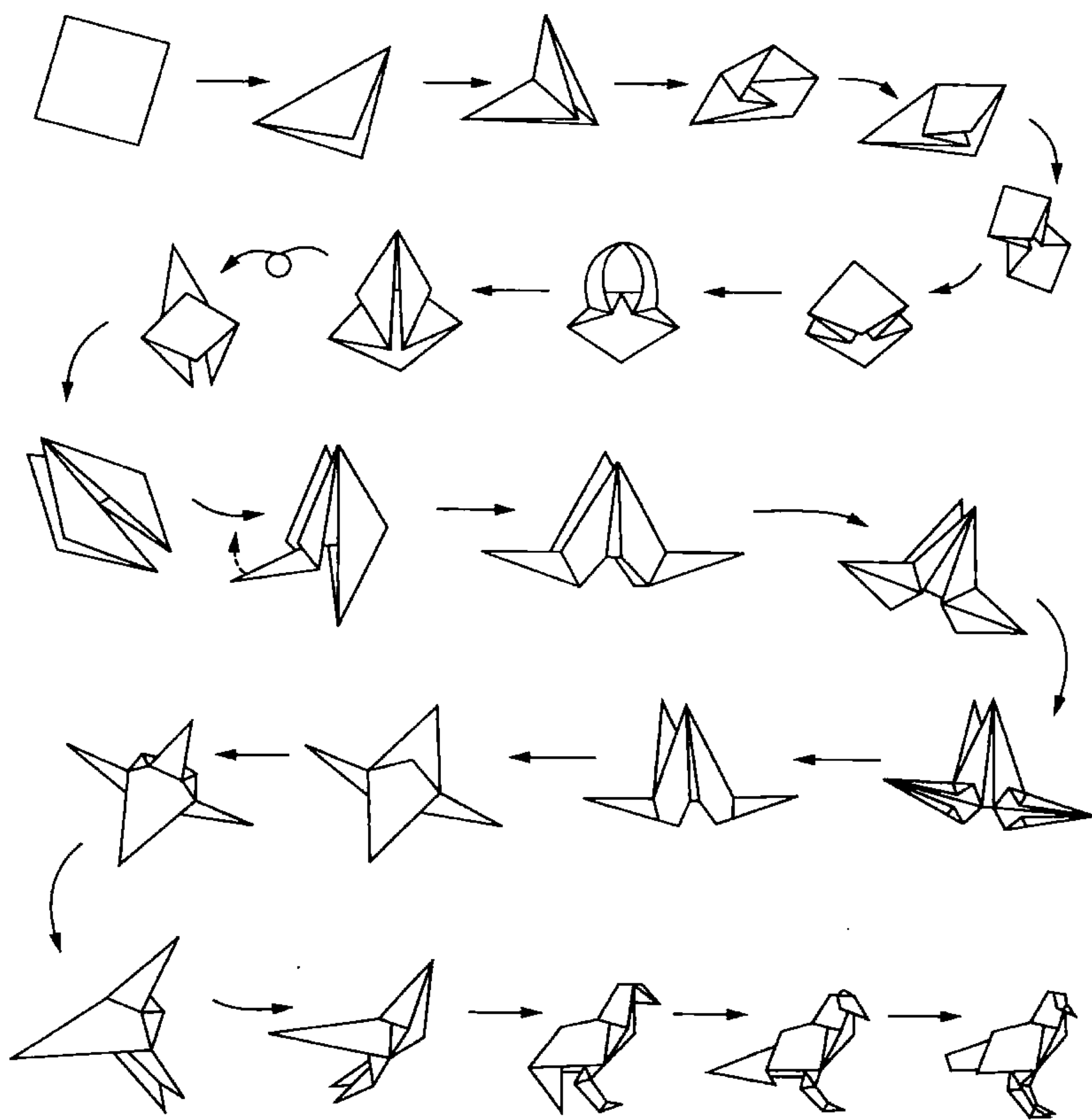


图5-2 由一张正方形的纸折叠成一只鸟

计算机科学解决这些问题的途径是建立一组严格定义的构建块，利用它们来构建算法的表示。这种构建块称作**原语**（primitive）。赋予原语准确的定义消除了很多由于歧义造成的问题，并且要求按照这些原语来描述算法就是确定了一致的细致程度。原语的集合以及说明如何组合这些原语来表示比较复杂的想法的规则集合就构成了一种**程序设计语言**。

每个原语都有自己的语法和语义。语法是原语的符号表示，语义是指该原语的含义。air 一词的语法由 3 个符号组成，然而其语义是一种充满整个世界的气体物质。作为一个例子，图 5-3 描述了折纸术中使用的一些原语。

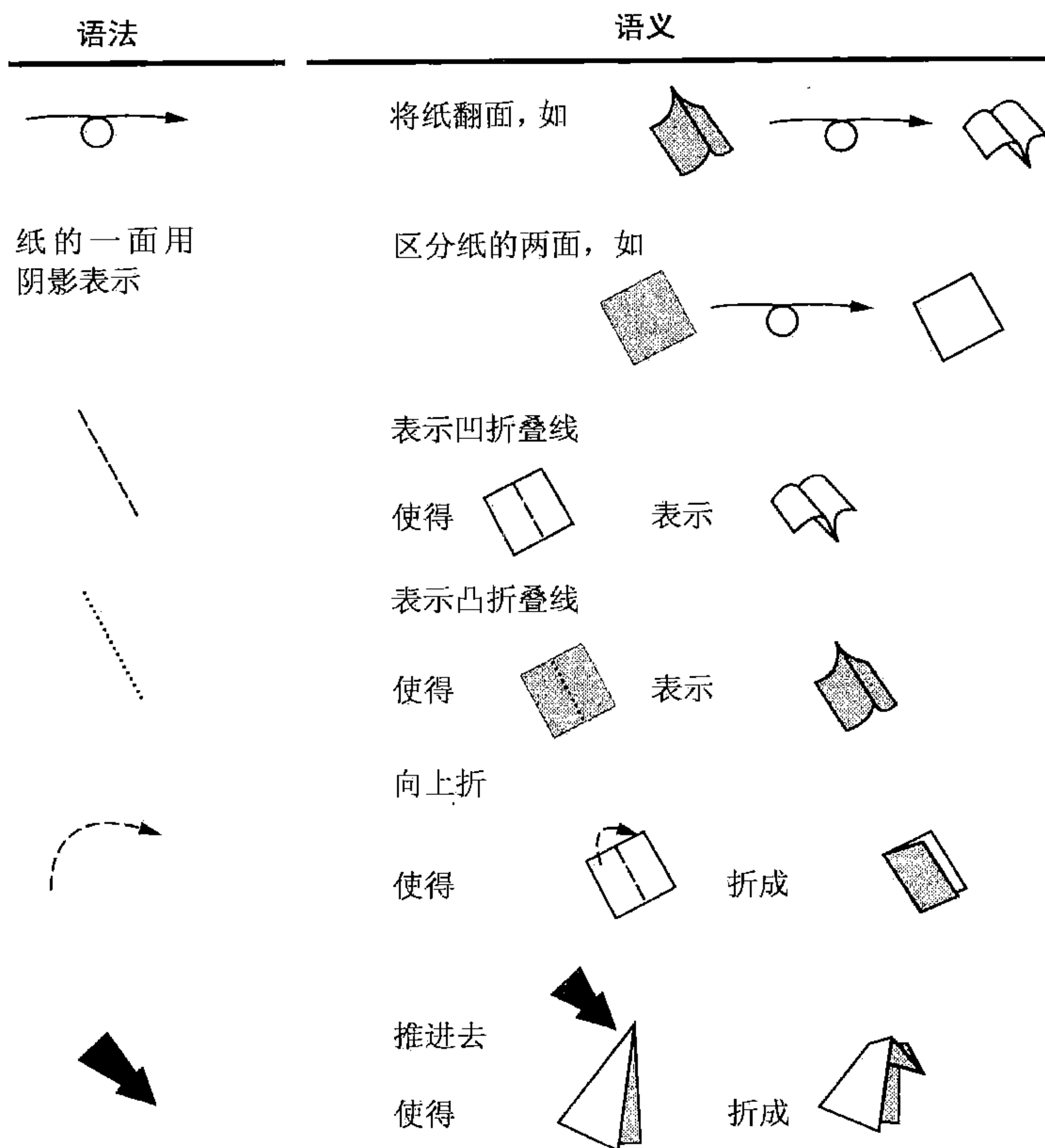


图5-3 折纸术的原语

为了获得用来描述由计算机执行的算法的原语的集合，我们需要借助于计算机执行的单个指令。如果一个算法在这个级别上加以描述，我们肯定会得到一个适合计算机执行的程序。然而，在这个级别上描述的算法是非常单调的，所以我们一般使用一些“更高级”的原语，其中每个原语都是由机器语言提供的较低级的原语组成的抽象工具。因此，我们可以得到一个概念上比机器语言更高级的方式来描述算法的正式的程序设计语言。我们将在下一章中讨论这种程序设计语言。

算法设计中的算法表示

设计一个复杂算法的任务要求设计者了解众多不相关的概念，这种需求可能超过人脑的能力。因此，复杂算法的设计者需要一种记录和重视不断发展的算法的部分的方法。

在20世纪50年代和60年代，流程图（算法可以由箭头相互连接的几何图形表示）表述了一种工艺状态的设计工具。然而，流程图经常变成一个由相互交叉的箭头组成的复杂网，这样就给理解算法的根本结构带来了难度。因此，流程图这种设计工具只能让位于其他方法。一个例子就是本书中使用的伪代码，通过这种伪代码，算法可以表示成定义明确的文本结构。另一方面，当我们的目标是表示算法而非设计的时候，流程图仍旧是有意义的。例如，图5-8和图5-9用流程图来展现出由流行的控制语句表示的算法结构。

对于更优的设计方法的研究仍在继续。在第7章中，我们将看到使用图形技术来辅助大型软件系统全球性设计的趋势；但是伪代码在设计系统中小一些的过程化构件中仍然很流行。

5.2.2 伪代码

现在我们暂时不介绍正式的程序设计语言，而是介绍一种非正式但更加直观的符号系统，这个系统称作伪代码。一般而言，**伪代码**（pseudocode）是一种在算法开发过程中非正式地表达思想的符号系统。

一种简单地获得伪代码的方法是放松正式语言用于表达最终算法的那些规则要求。这个方法通常是在已经预先知道目标程序设计语言的情况下使用。这种在程序开发初期使用的伪代码是由语法-语义结构组成的，这种结构类似于目标程序设计语言所使用的结构，但是不那么正规。

当然，我们的目标是在不把讨论限定于特定程序设计语言的情况下考虑算法的开发和表示问题。因此，我们得到伪代码的办法是开发一种一致且简明的用来表示循环语义结构的表示法。这样一来，这些结构将成为我们表达思想的原语。

这样的循环语义结构之一是保存计算的值。比如，我们计算了日用账户和存款账户上的结余总和，并打算保存这个结果以便以后引用。在这种情况下，我们用：

名字←*表达式*

形式表达，其中，*名字*是我们欲引用结果的名字，而*表达式*则描述其结果将被保存的计算。我们将这个语句读为“把表达式的值赋给该名字”，并且称该语句为**赋值语句**（assignment statement）。例如，语句

```
RemainingFunds ← CheckingBalance + SavingsBalance
```

是把CheckingBalance与SavingsBalance的值相加的结果赋给名字RemainingFunds的赋值语句。这样，RemainingFunds可以在将来的语句中用于引用该总和的值。

另一个递归语义结构是，根据某个条件的真与假从两个可能的活动中选择一个。这样的例子有：

如果国内生产总值增长了，那么买进普通股票；否则，卖出普通股票。
若国民生产总值增长了则买进普通股票，否则就卖出。
买进或卖出普通股票取决于国内生产总值的增长或减少。

其中每个语句都可以写成符合如下结构的形式：

```
if (条件) then (活动)
        else (活动)
```

这里，我们使用了关键字if（如果）、then（那么）和else（否则）来指示这个主结构中的不同子结构，并且使用括号来限定子结构的界限。采用这种语法结构作为伪代码，我们便得到了可以表达这种常用语义结构的统一方法。因此，尽管语句：

根据该年份是否是闰年，总天数相应地被366或365除。

可能会产生一种更富有创造性的文字风格，但是我们将坚持选择简单的形式：

```
if (年份是闰年)
  then (总天数 ← 总天数被366除)
  else (总天数 ← 总天数被365除)
```

对于不涉及else活动的情况，我们也可以采用较短的语法：

```
if (条件) then (活动)
```

利用这种表示形式，语句

如果处于销售额减少的场合，那么价格降低5%。

将可以简化为：

```
if (销售额降低) then (价格降低5%)
```

另一个常用的语义结构是，只要某个条件为真，一个语句或一组语句将重复地执行。这样的例子有：

只要有票可卖，那么继续售票。

和

当有票可以卖时，保持售票的状态。

对于这两种情况，我们采用下列统一的模式作为伪代码：

```
while (条件) do (活动)
```

简而言之，这个语句意味着检查条件，如果为真，那么实现*活动*，然后返回再次检查条件。但是，如果发现条件为假，那么就转到while结构后的下一个指令。于是，前面的两个语句可以简化为：

```
while (仍然有票可卖) do (卖票)
```

缩进通常可以提高程序的可读性。例如，语句

```
if (未下雨)
  then (if (温度=热)
            then (去游泳)
             else (去打高尔夫)
          )
  else (看电视)
```

比该语句的下述格式容易理解：

```
if (未下雨) then (if (温度=热) then (去游泳)
else (去打高尔夫)) else (看电视)
```

所以，在我们的伪代码中将使用缩进。（注意，我们甚至可以利用缩进来调整闭括号的位置，使得它与对应的开括号对齐，以辨认语句或短语的作用范围。）

我们想用伪代码来描述那些在其他应用中可能作为抽象工具的活动。对于这样的程序单元，计算机科学有许多术语，如子程序、子例程、过程、模块和函数等，其中每一个在含义上都有自己的变化。对于伪代码，我们将采纳过程（procedure）这个术语，并利用这个术语来给出一个标题，作为这个伪代码单元的名字。更精确地说，我们将下列形式的语句作为一个伪代码单元的开始：

```
procedure 名称
```

其中，*名称*是该单元特有的名字。在这个引导性语句的后面是一系列定义该单元动作的语句。例如，图5-4是称为Greetings的过程的伪代码表示，该过程打印“Hello”3次。

```
procedure Greetings
Count ← 3;
while (Count > 0) do
    (打印信息 "Hello";并且
    Count ← Count - 1)
```

图5-4 伪代码形式的过程Greetings

当一个过程所实现的任务在伪代码其他地方需要时，只需要通过名称来请求它。例如，如果有取名为 ProcessLoan 和 RejectApplication 的两个过程，那么可以通过下面的语句在 if-then-else 结构内请求它们的服务：

```
if (...) then (执行过程ProcessLoan)
          else (执行过程RejectApplication)
```

当被测试的条件为真时，执行过程 ProcessLoan，而在条件为假时，执行过程 RejectApplication。

如果过程用于不同的环境，设计伪代码时应该使其尽可能通用。一个给名字列表排序的过程应该设计成能够给任何列表（而不是特定的列表）排序，因此应该按照这样的要求来编写该过程，即要排序的列表不在过程内部指定。事实上，该列表应在这个过程的伪代码里以类属名来指称。

在伪代码里，我们将采用这样的约定：这些类属名（称为**参数**）列在括号里，并在标识过程名字的同一行上。例如，一个名为 Sort 的过程（设计成对任何名字列表进行排序）以下列语句开始：

```
procedure Sort (List)
```

在后面的伪代码里，在需要引用要排序的列表时，就可以使用类属名 List。同样，当需要 Sort 服务时，我们将知道是什么列表要替代过程 Sort 的参数 List。于是根据需要，可以写成：

```
把过程Sort应用在机构成员列表
```

和

```
把过程Sort应用在婚礼来宾列表
```

命名程序中的项

在自然语言中，项通常使用多个单词的名称，例如："cost of producing a widget" 或 "estimated arrival time"。经验表明，在一个算法表示中使用这种多个单词的名称会使算法的描述复杂化。最好让每个项使用一个连续的文本块指示。多年来，许多技术被用于将多个单词紧缩为单个词法单元，以便为程序中的项提供描述性名称。一种技术是使用下划线来连接单词，建立诸如estimated_arrival_time这样的名称。另一种技术是用大写字母帮助读者理解紧缩的多单词名称。例如，我们可以对每个单词的首字母使用大写，获得诸如EstimatedArrivalTime这样的名称。这种技术称为**Pascal样式**（Pascal casing），因为该样式在Pascal程序设计语言的用户中很流行。一种Pascal样式的变形称为**camel样式**（camel casing），该样式除了首字母小写外，其他与Pascal样式相同，如estimatedArrivalTime。本书中，我们使用Pascal样式，但这种选择很大程度上是个人偏好。

记住，伪代码的目的是要提供一种用可读的、非形式的方法来表示算法。我们希望符号系统能够帮助我们表达思想，而不受制于严格的形式规则。因此，在需要的时候，我们可以随意扩充或修改伪代码。特别是，如果一组括号中的语句又涉及带括号的语句，会使括号配对很困难。在这些情况下，许多人发现在闭括号后面加上简短的注释，说明是哪个语句终止了，是非常有帮助的。特别是，在 while 语句的闭括号后面可以加上 end while，产生下列形式的语句：

```
while (...) do
  ( .
    .
    .
  ) end while
```

或者

```
while (...) do
  (if (...)
    then ( .
           .
           .
         ) end if
  ) end while
```

其中我们已经指明了 if 和 while 语句的结束。

我们的目的是用一种可读的形式来表达算法，因此可以随时引进一些直观的辅助工具（缩进、注释等）来达到这个目的。其次，如果碰到了一个尚未在我们的伪代码中体现的递归问题，那么可以选择扩充伪代码，采用一致的语法来表示这个新的概念。

问题与练习

1. 一种环境下的原语可以是另外一种环境下原语的“合成物”。比如，while语句是伪代码中的一个原语，但是它被作为机器语言指令的一个“合成物”加以实现。给出计算机以外的这种现象的两个例子。
2. 从何种意义上讲，过程的结构就是原语的结构？
3. 欧几里得算法给出了求两个正整数X和Y的最大公约数的算法：只要X和Y的值均不是0，则继续用较大的数除以较小的数，并且将除数和余数分别赋给X和Y（X最后的值就是最大公约数）。用伪代码的方式表达这个算法。
4. 描述一个在计算机程序设计以外的学科里使用的原语集合。

5.3 算法的发现

程序开发由两个活动组成——发现潜在的算法和以程序的方式表示算法。从这点看，我们一直在关注算法表示的问题而未把算法是如何被发现的问题放在首位，但是算法的发现在软件开发过程中往往是更加具有挑战性的步骤。毕竟，发现一个算法来解决问题需要找到一个解决该问题的方法。因此，要理解算法是如何发现的就是要理解问题的求解过程。

5.3.1 问题求解的艺术

问题求解的技术和学习更多相关知识的需求并不只存在于计算机科学中，这是一个几乎在任何领域中都永久存在的问题。由于算法发现的过程和一般问题的求解过程之间存在着紧密的联系，使得计算机科学进入了那些试图寻找更好的问题求解方法的学科中。最终，我们希望可以把问题的求解简化为一个算法，但是已经证明这是不可能的。（这是第 12 章相关内容的结果，

在第 12 章我们将展示有些问题是找不到算法解决方法的。）因此问题求解能力更多地成为一种有待开发的技艺，而非需要学习的精确科学。

作为问题求解难以琢磨、颇具艺术性的本质的证据，数学家波利亚（G. Polya）在 1945 年列出了以下非严格定义的问题求解阶段，其蕴涵的基本原理直至今日仍然是很多人教授问题求解技能的基础。

第1阶段　理解问题。

第2阶段　设计一个解决问题的计划。

第3阶段　完成计划。

第4阶段　从准确度及其是否有潜力作为一个解决其他问题的工具这两方面来评估这个计划。

我们把上述阶段移植到程序开发的语境中，这些阶段变成：

第1阶段　理解问题。

第2阶段　寻找一个可能解决问题的算法过程的思路。

第3阶段　阐明算法并且用程序将其表达出来。

第4阶段　从准确度及其是否有潜力作为一个工具解决其他问题这两方面来评估这个程序。

在描述完波利亚的观点后，我们应该着重强调这些阶段并不是在尝试求解问题的时候需要遵循的步骤，而是在求解过程中有时需要完成的阶段。这里的关键词是“遵循”。仅遵循这些步骤是不能求解问题的，要求解问题，必须有创新精神和领先一步的意识。如果在求解问题的时候总是抱有“现在我完成了第 1 阶段，该是开始第 2 阶段的时候了”之类的想法，那么你可能根本不能成功。然而，如果仔细考虑问题并且最终解决了它，可以回想在解决问题的过程中你都做了些什么，并且可以发现确实经历了波利亚所描述的各个阶段。

波利亚阶段原理的另外一个重要观点是并不一定要按顺序执行这些步骤。成功的求解问题者通常是在完全理解问题本身（阶段 1）之前就开始设计构想解决问题的策略（第 2 阶段）。然后，如果他们的策略失败了（在第 3 阶段或者第 4 阶段），这些人会对这个问题的复杂程度有更深的理解。基于这些比较深入的理解，他们会回过头去构想另一个更有希望成功的策略。

必须记住的是，我们正在讨论怎样求解问题——并不是我们希望问题如何解决。在理想情况下，我们希望消除前面描述的“尝试-错误”过程中的固有的浪费。在开发大型软件系统的情况下，如果在像第 4 阶段这样晚的时候才发现问题，那么就会导致资源的极大浪费。避免这样的灾难是软件工程师的主要目标（第 7 章），他们习惯于在寻求解决方案之前，坚持对该问题有一个全面透彻的理解。当然，有些人可能会说，在一个问题解决之前是不可能真正理解这个问题的。起码事实上，问题无法解决表明缺乏对问题的理解。因此，坚持在提出任何解决方案之前必须对问题有完全理解的想法看起来有些过于理想化。

作为一个例子，我们考察下列问题：

> 甲承担了确认乙的3个孩子的年龄的任务。乙告诉甲3个孩子的年龄乘积是36。在考虑了这个线索以后，甲要求乙给出另外的线索，于是乙告诉甲3个孩子的年龄之和。甲再次要求乙给出其他线索，乙告诉甲他的最大的一个孩子弹钢琴，在得到这个线索之后，甲得到了乙的3个孩子的年龄。
>
> 乙的3个孩子的年龄分别是多少？

乍一看，最后一个线索与问题完全没有关系。但是显然，正是因为这条线索，甲最后确定了 3 个孩子的年龄。这是为什么呢？让我们制订一个计划并且遵循这个计划，尽管我们对于这个问题还有很多疑问。我们的计划是跟踪问题陈述所描述的步骤，同时在这个进程中记录对甲有用的信息。

第一条线索告诉甲，3 个孩子的年龄之积是 36。这意味着表述 3 个年龄数值的三元组肯定是图 5-5a 中列出的三元组之一，第二条线索是期望的三元组内 3 个数字之和。我们并不知道这个和到底是多少，但是知道这个信息并不足以让甲得到正确的三元组，所以期望得到的三元组的和在图 5-5b 中的表里面至少出现两次。此处只有(1, 6, 6)和(2, 2, 9)具有相同的和，这两组数字的和均是 13。当给出最后一条线索的时候，我们最终理解了最后一条线索的重要性。这个信息与弹钢琴本身没有什么关系，而是说明了只有一个孩子年龄最大的事实。这条线索将三元组(1, 6, 6)排除并且最终得到结论，就是 3 个孩子的年龄分别是 2、2 和 9。

(a) 乘积为36的三元组		(b) (a)中每个三元组的和	
(1,1,36)	(1,6,6)	1 + 1 + 36 = 38	1 + 6 + 6 = 13
(1,2,18)	(2,2,9)	1 + 2 + 18 = 21	2 + 2 + 9 = 13
(1,3,12)	(2,3,6)	1 + 3 + 12 = 16	2 + 3 + 6 = 11
(1,4,9)	(3,3,4)	1 + 4 + 9 = 14	3 + 3 + 4 = 10

图5-5 计算过程

在这个例子中，直到我们尝试实施解决问题的计划（第 3 阶段）的时候，才获得对这个问题的完全理解（第 1 阶段）。如果坚持要首先完成第 1 阶段，我们可能根本得不到问题的答案。这种解决问题过程中的不规则性是开发问题求解的系统方法的基础。

另外一个不规则性在于，那些还没有得到明显成功的问题解决者可能在完成其他任务的时候突然得到的神奇灵感，并发现原来问题的一个解决方法。H. von Helmholtz 早在 1896 年就发现了这种现象，并且数学家庞加莱（Henri Poincaré）在巴黎对心理学会的一次演讲中对此进行了讨论。在这个演讲中，庞加莱叙述了他在解决一个问题时的经历：他将原来的问题放在一边儿，开始做其他工作之后，却突然意识到原来问题的一个解决方法。这种现象反映出这样一个过程，大脑的潜意识部分好像一直在思考问题，如果成功，便会把解决方法反映给大脑的有意识部分。今天，我们把在对于问题的有意识的工作与突然的灵感之间的这个时期称作沉思期，对于这个时期的理解仍旧是当前研究的目标。

5.3.2 入门

前面，我们已经从一些心理学的观点讨论了问题求解，但是回避了直接对质这样的一个问题，即应该如何求解问题。当然有很多问题求解的方法，每一种方法都可能在某些场合获得成功。我们将简要地介绍其中一些方法。目前，我们注意到这些技术中贯穿着一条普遍的线索，简单地说就是要“入门”。作为一个例子，让我们考虑下面这个简单问题。

> 在甲、乙、丙和丁进行赛跑之前，他们分别对结果进行了预测：
>
> 甲预测乙将会获胜；
>
> 乙预测丁将是最后一名；
>
> 丙预测甲是第三名；
>
> 丁预测甲的预测将是正确的。
>
> 这几个预测只有一个是正确的，并且是最后的获胜者作出的预测，请据此给出甲、乙、丙、丁赛跑的名次排序。

在阅读了这个问题并且对数据进行分析之后，我们很快就可以认识到，因为甲和丁的预测是等价的，而只有一个人的预测正确，所以这两个预测都是错误的。因此甲和丁都不是胜利者。在这一点上我们已经为解决这个问题迈出了第一步，并且发现获得完整的解决方法的过程仅仅

是以此为基础进一步扩展知识。如果甲的预测错误，那么乙也不是胜利者。这样就只剩下了一个选择，就是丙是胜利者。因此，丙的预测是正确的。从而，我们知道甲是第三名。这就意味着最后的比赛名次是丙、乙、甲、丁，或者丙、丁、甲、乙，但是前者被排除了，因为乙的预测肯定是错误的。因此最后的顺序是：丙、丁、甲、乙。

当然，知道怎样进入问题并不等于知道如何去做这件事。得到立足点，并认识到如何把对于问题的初始介入扩展为问题的解决方法，要求问题解决者有创意。对于如何入门波利亚和其他人提出了很多通用的方法，其中一个是反方向解决问题。比如，如果问题是找到对于一个已知输入产生一个特定输出的方法，我们可以从输出开始，然后反向推导出输入。这个方法就是人们在解决本节前面的学习叠纸鸟的基本思路。我们把一个已经完成的纸鸟拆开来，然后看看如何能将它折好就可以了。

另一个通用的解决问题的方法是寻找一个相关的、解决起来较简单的并且在此以前已经得到解决的问题，然后尝试把这个解决方法用到当前问题中。这个技术在程序开发中特别有用。通常，程序开发并不是解决一个问题的一个特定实例，而是要寻找一种适用于求解一个问题的所有实例的一般算法。更加准确地讲，如果面对一个要把姓名列表按照字母排序的程序开发任务，我们的任务并不是只给一个特定的列表排序，而是寻找一个可以用来给任何名单排序的通用算法。因此，尽管指令

交换名字David和Alice。
将名字Carol移到Alice和David之间
将名字Bob移到Alice和Carol之间。

可以把由David、Alice、Carol和Bob组成的名单正确排序，但这并不是我们需要的通用的算法。我们需要的算法应当既可以为这个名单排序，又可以为其他名单排序。这并不是说为特定列表排序的算法在我们研究通用算法的过程中是完全没有意义的。例如，我们可以通过考虑特殊的情况来进入问题，来寻求能够用于开发通用算法的一般原则。于是，在这种情况中，解决问题的方法可以从对于多个相关问题的解决中得到。

另外一个进入问题的方法是**逐步求精**（stepwise refinement），这种方法本质上不是试图立即解决整个问题，而是首先把一个手头的问题看做多个子问题。我们可以按照步骤通过解决各个子问题来最后解决整个问题，其中每一步都比解决完整的问题要更容易。逐步求精的方法还可以把这些步骤划分成更小的步骤，然后这些更小的步骤还可以继续进行划分，直到整个问题被简化为一组简单的子问题为止。

从这点看来，逐步求精是一种**自顶向下方法**（top-down methodology），这种方法从一般发展到特殊。相反，**自底向上方法**（bottom-up methodology）是从特殊发展到一般。尽管理论上相反，但是实际上这两种方法在应用中互为补充。比如，逐步求精的自顶向下方法分解问题通常是由那些可能从事自底向上工作的解决问题的人指导的。

逐步求精的自顶向下方法从本质上讲是一种组织工具，这种工具解决问题的属性是组织方式的结果。逐步求精早已成为数据处理中一个重要的设计方法，其中大型软件系统开发项目拥有一个很大的组织模块。但就像我们将要在第7章中学习的，大的软件系统越来越多地通过结合预先编制的部件完成（本质上是一种自底向上的方法），因此自顶向下和自底向上的方法仍然是计算机科学中重要的工具。

之所以维持如此宽广的观点是基于以下的一个事实，即将事先形成的观念和预选的工具带入问题求解任务中，有时候可能掩盖问题的简单性。本节前面讲的求解3个孩子年龄的问题就是这种现象的一个很好的例子，学习代数的学生解决问题时总是给出系统的联立方程，这种方法可能将问题带入死路，而且常使问题解决者误认为并没有足够的信息来解决问题。

另外再给出一个类似的例子：

> 当你从码头走上船的时候，帽子掉进了水里，但是你并不知道。河水的流速是2.5英里[1]/小时，所以帽子开始向下游漂去。同时，你开始以相对于水流4.75英里/小时的速度向上游前进。10分钟后，你发现帽子不见了，然后调转船头，开始追你的帽子。试问多长时间可以找到帽子？

大多数学习代数的学生还有那些热衷使用计算器的人在解决这个问题的时候，首先会确定船在10分钟后向上游行进了多远以及在相同时间内帽子向下游漂了多远。然后，他们确定船将使用多少时间赶上帽子。但是，当船到达这个位置的时候，帽子又向下游漂了一段距离。因此，问题解决者要么就是用微分的方法重新解题，要么就陷入到计算每次船到达帽子的上一个位置的时候帽子所处的位置这样一个怪圈里。

其实问题很简单。这里的失误在于解题者忙于列出公式并进行求解。其实，我们需要先将技术放在一边，并且调整对于问题的观点。整个问题发生在河中。事实是水相对于河岸的流动与解题是不相关的。想象一个相同的问题发生在传送带上而非水中。首先，在传送带停止的情况下解决问题。如果你站在传送带上，然后把帽子放在脚下，之后反向行走10分钟，那么返回到帽子所在处需10分钟。现在启动传送带，这意味着旁边的场景将相对于传送带反向运动起来。但是，因为你站在传送带上，这就不会改变你和传送带或者帽子之间的相对关系，所以仍会使用10分钟回到放帽子的地方。

我们可以得到这样一个结论，算法的发现仍旧是一种富有挑战的艺术，这项工作必须花费一定时间才可以完成，而不能像一门由明确的方法组成的学科那样学到。因此，机械地训练未来的问题解决者使之遵循一定的方法，就是在压制那些本来应该被培养出来的创造性技能。

问题与练习

1. a. 寻找一个算法求解下面的问题：已知一个正整数n，寻找一个正整数列表，该列表中所有正整数的乘积是其正整数和为n的所有正整数列表中最大的。例如，如果n为4，那么所求的列表由两个2组成，因为2×2大于1×1×1×1、1×1×2和1×3。如果n为5，则所求的列表由2和3组成。
   b. 如果n=2001，那么所求的列表由哪些数字组成？
   c. 说明你如何“入门”这个问题的。
2. a. 假设已知跳棋棋盘由2^n行和每行2^n列组成，给定正整数n，以及一盒L型棋子，每一个都恰好可以覆盖棋盘的3个正方形格子。如果任何一个格子从棋盘上被切掉，我们是否还可以用这些棋子在既不互相重叠，又不跨越棋盘边的条件下把剩余的棋盘填满？
   b. 请说明怎样用问题a的解来证明：对于所有的正整数n，$2^{2n}-1$是可以被3除尽的。
   c. 说明问题a和问题b与波利亚的问题求解步骤的关联性。
3. 解码下面的消息，并解释你是如何入门的。
   Pdeo eo pda yknnayp wjosan.
4. 如果打算解决拼图游戏问题，即把图片抛散在桌面上然后将其拼出完整图，你会采用自顶向下的方法吗？如果是看着拼图盒上的完整图来做，你的回答会改变吗？

5.4 迭代结构

我们现在要学习一些在描述算法过程中使用的重复结构。在本节中，我们讨论**迭代结构**

[1] 1英里=1.6093千米。——编者注

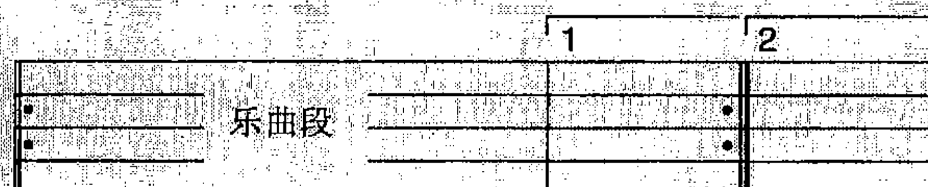
（iterative structure）。在这种结构中，一组指令以循环方式重复执行。在5.5节中，我们将介绍递归技术。作为一些关联知识，我们将介绍一些流行的算法——顺序搜索法、二分搜索法和插入排序法。我们从介绍顺序搜索法开始。

音乐中的迭代结构

音乐家比计算机科学家早几个世纪就已经开始使用和编写迭代结构。实际上，一首歌的结构（谱写成多个小节，每个小节配以和声）可以用while语句示范：

```
while（有剩余小节）do
    （唱下一小节；
     唱和声）
```

此外，下面的乐谱



只不过是谱曲者表达下面结构的方式：

```
N←1;
while （N<3） do
    （演奏该乐曲段;
     演奏第N段尾曲;
     N←N+1）
```

5.4.1 顺序搜索法

考虑一下查找某个特定值是否存在于一个列表中的问题。我们希望开发一个算法来确定这个值是否在列表中。如果它在列表中，我们认为查找成功，反之则认为查找失败。我们假设被查找的列表依照某种规定已被排序。例如，如果这是一个姓名列表，我们假设列表中的名字是按照字母顺序排列的；如果这个列表由数字组成，我们假设里面的表项按照增序排列。

为了入门，我们想象如何在一个大概有20条记录的来宾列表中寻找一个特定的姓名。在这种情况中，我们可以从头开始扫描整个列表，将每一条记录与目标姓名进行比较。如果找到了目标姓名，那么查找就将以成功终止。当然，如果到达列表的最后仍然没有找到目标，查找就以失败告终。实际上，如果到达了（从字母顺序方面看）大于目标姓名的值还没有找到目标姓名，那么我们的查找就已经宣告失败。（记住，列表已经按照字母顺序排列，所以到达一个大于目标姓名的表项就意味着目标不可能在列表中出现了。）概括来说，我们粗略的想法是只要还有姓名没检查而且目标姓名大于正在检查的表项，查询就将继续。

在伪代码中，这个过程可以表达为：

```
选择列表中的第一个表项作为TestEntry
while（TargetValue>TestEntry并且还有表项没有检查 ）
    do（ 选择列表中下一个表项作为TestEntry）
```

如果要终止上面的 while 结构，则两个条件中有一个必须为真：或者目标值被找到，或者目标

值不在列表中。在任何一种情况中，我们都可以通过比较测试表项（TestEntry）和目标值来发现查找是否成功。如果这两个值相等，那么查找就成功了。因此，我们把下面一些语句添加到以上伪代码例程的下面：

```
if ( TargetValue = TestEntry )
  then ( 宣布查找成功 )
  else ( 宣布查找失败 )
```

最后，我们观察这个例程的第一条语句。这条语句把列表中的第一条记录选做测试表项，这种选择基于表中至少有一条记录这样的假设。我们可能认为这是一种安全的猜测，但是为了保险起见，我们将前面的例程作为下面语句中的 else 部分：

```
if ( 列表空 )
  then ( 宣布查找失败 )
  else ( . . . )
```

这个过程的伪代码如图 5-6 所示。注意，这个过程可以在其他过程中通过下面的语句加以使用：

```
运用过程Search于旅客列表查找名为Darrel Baker的旅客
```

这个语句可以查明 Darrel Baker 是否是一名旅客，而若用语句：

```
利用nutmeg作为目标值运用过程Search于原料列表
```

则可以查明 nutmeg（肉豆蔻）是否出现在原料列表上。

```
procedure Search ( List,TargetValue )
if ( List空 )
   then
     ( 宣布查找失败 )
   else
     ( 选择列表中的第一个表项作为TestEntry;
     while ( TargetValue > TestEntry 并且还有表项没有检查 )
       do ( 选择列表中下一个表项作为TestEntry ) ;
     if ( TargetValue = TestEntry )
         then ( 宣布查找成功 )
         else ( 宣布查找失败 )
     ) end if
```

图5-6 伪代码形式的顺序搜索算法

概括来讲，图 5-6 所示的算法按照表项在列表中的出现顺序进行查找。因此，这个算法称作**顺序搜索**（sequential search）算法。因为其简单，所以顺序搜索法经常用于在较短的列表中进行查找或因其他方面的考虑需要使用它的情况。在比较长的列表中，顺序搜索就没有（我们将要学习的）其他技术有效了。

5.4.2 循环控制

一条指令或者一系列指令的重复使用是一个很重要的算法概念。一种实现这种重复的方法是称作**循环**（loop）的迭代结构。这种结构中，一组称为循环体的指令在某些控制过程的指引

下重复执行。一个典型的例子就是图 5-6 所示的顺序搜索算法。这里我们使用 while 语句来控制以下单条语句的循环：

```
选择列表中下一表项作为TestEntry
```

实际上，while 语句：

```
while ( 条件 ) do ( 循环体 )
```

就是循环结构的一个例子，它的执行所跟踪的循环模式为：

```
检查条件，
执行循环体，
检查条件，
执行循环体，
……
检查条件。
```

直到条件为假。

作为一个普遍规则，循环结构的使用使程序得到了比仅仅将循环体重写多次更高的灵活度。例如，执行下面的语句 3 次：

```
加一滴硫酸
```

等价于语句序列

```
加一滴硫酸，
加一滴硫酸，
加一滴硫酸。
```

但是我们写不出与下面的循环结构等价的具有相似结构的序列：

```
while ( pH值大于4 ) do
     ( 加一滴硫酸 )
```

因为我们事先不知道需要滴入多少硫酸才合适。

现在让我们进一步考查循环控制的组成。你可能认为这一段关于循环结构的部分并不重要。毕竟，通常都是循环体在实际执行手头的任务（比如，加几滴硫酸）——控制活动看起来仅是相关的开销，因为我们选择在重复的形式中执行循环体。但是，经验表明循环控制是循环结构中一个非常容易出现错误的部分，所以很值得我们注意。

循环控制由初始化、测试和修改（如图 5-7 所示）3 个活动组成，其中每一个活动都决定了循环控制是否能够成功。测试活动有责任通过查看表明应终止的条件来终止循环过程。这个条件就是**终止条件**（termination condition）。为了这个测试活动，我们在伪代码的每一个 while 语句中都提供一个条件。在 while 语句中，循环体必须在阐明的条件下执行——终止条件就是 while 结构中出现的条件的对立条件。因此，在语句

```
while ( pH值大于4 ) do
    ( 加一滴硫酸 )
```

初始化：	设置一个初始状态，这一状态将被修改直至终止条件
测　试：	比较当前状态和终止条件，如果相等则终止循环
修　改：	修改状态使之可以达到终止条件

图5-7　可重复结构控制的组成

中，终止条件是“pH值不大于4”，在图5-6所示的while语句中，终止条件是：

（目标值≤测试项）或（不再有要检查的表项）

循环控制中的另外两个活动确保了终止条件最终可以出现。初始化步骤建立了一个开始条件，修改步骤将这个条件移向终止条件。比如，在图 5-6 中，初始化发生在 while 语句之前的语句中，该处当前的测试表项被设定为列表的第一条记录。在这个例子中，修改步骤实际上是在循环体内完成的，在循环体中，我们将测试位置（也就是测试表项）向列表的末尾移动。因此，在执行完初始化步骤后，修改步骤的重复执行最终使得程序可以到达终止条件。（或者到达一条大于或等于目标值的测试项，或者到达列表的末尾。）

我们应该强调的是初始化和修改步骤必须导致合适的终止条件。这个特性对于正确的循环控制至关重要，因此在设计循环结构的时候必须仔细检查它是否存在。如果没有进行相应的检查，在最简单的例子中都会发生错误。一个典型的例子就是：

```
Number ← 1;
while (Number ≠ 6) do
  (Number ← Number +2)
```

这里，终止条件是“Number=6”。但是 Number 初始化为 1，并且每次修改的时候都加 2。因此，在循环过程中 Number 的值将是 1、3、5、7、9 等，但是永远不会是 6，这样，循环就无法终止。

循环控制部件的执行次序可产生微妙的结果。事实上，有两种常用的循环结构，它们仅仅在循环控制部件执行次序上有所区别。第一个如以下伪代码语句所示：

```
while ( 条件 ) do ( 活动 )
```

这个结构的语义由图 5-8 中的**流程图**（flowchart）给出。（流程图使用各种形状来表达单个步骤并且用箭头表达步骤的顺序。线框形状之间的不同表示相关步骤涉及的动作类型不同，比如菱形表示判断，而矩形表示任意语句或语句序列。）注意，while 结构的终止测试出现在循环体执行之前。

相反，图 5-9 中所示的结构要求循环体在终止条件测试之前执行。在这种情况下，循环体总是至少执行一次，然而在 while 结构中，如果终止条件在第一次测试时就满足，则循环体一次都不用执行。

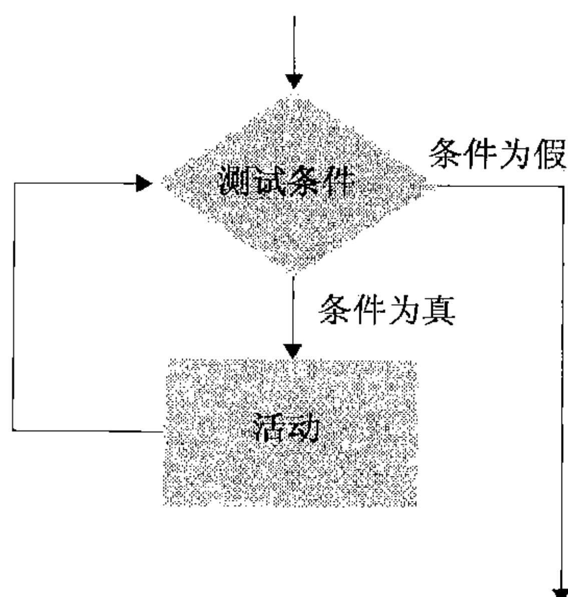


图5-8　while循环结构

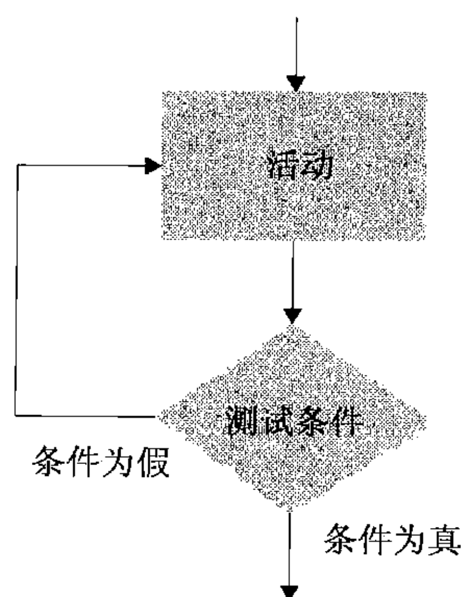


图5-9　repeat循环结构

我们用语法格式：

```
repeat ( 活动 ) until ( 条件 )
```

在伪代码中表示图 5-9 所示的结构。因此，语句：

```
repeat ( 从你口袋里面取出一个硬币 )
until ( 你口袋里没有硬币 )
```

假设开始时在你的口袋中至少有一个硬币，但是下面的语句：

```
while ( 你口袋里面有硬币 ) do
   ( 从你口袋里取出一个硬币 )
```

就不再限定必须至少有一枚硬币。

依照伪代码的术语，我们通常会把这些循环称作 while 循环结构或者 repeat 循环结构。在更一般的情况下，有时候我们用**前测试循环**（pretest loop）一词来表示 while 循环结构（因为终止条件的检查是在执行循环体之前执行的），而把 repeat 循环结构称作**后测试循环**（posttest loop）（因为终止条件的检查发生在执行循环体之后）。

5.4.3 插入排序算法

作为另外一个迭代结构的例子，下面考查将一个姓名列表按照字母顺序进行排序的问题。但是在继续介绍之前，我们应该了解排序的限制。简单来讲，我们的目标是在一个列表的内部将表项进行排序，即通过把表项移来移去来将列表排好序，而不是把列表移到其他位置。我们的情况类似于这样的列表排序问题，每个表项记录在一张索引卡片上，卡片分散在桌面上，把桌面挤得满满的。我们已经清理出足够的空间来放这些卡片，但是不允许挪开其他东西来挤出更多的空间。这种限制在计算机应用里是很典型的，其原因当然不是计算机里的工作空间一定像桌面那样拥挤，而仅仅是因为我们希望更有效地利用存储空间。

让我们从考虑如何在这样一个桌面上进行排序来“入门”。考虑一个名字列表：

Fred
Alex
Diana
Byron
Carol

一个方法是每次只对这个表中的一个子表进行排序。Fred 在表的最顶部，现在只看他和 Alex。因此我们只要拿出包含姓名 Alex 的卡片，将 Fred 放到它留下的空缺处，然后把 Alex 放到 Fred 原来的位置，如图 5-10 第一行所示。这样名字列表就变成了：

Alex
Fred
Diana
Byron
Carol

现在，顶部的两个名字构成了一个已经排序的子列表，但是最顶部的 3 个名字还不是。因此，我们可以取出第三个名字 Diana，然后将 Fred 移动到 Diana 拿走后留下的空白处，然后将 Diana 放入 Fred 原来所在的位置，如图 5-10 中第二行所示。最顶部 3 条记录已经排好序了。继续这个操作，通过取出第四个姓名 Byron，然后把 Fred 和 Diana 下移同时将 Byron 插入空缺处，就能够获得最顶部 4 个记录均被排序的列表了（参见图 5-10 的第三行）。最后，我们通过取出 Carol，然后把 Fred 和 Diana 下移，同时把 Carol 放入空缺来完成排序工作（参见图 5-10 的第四行）。

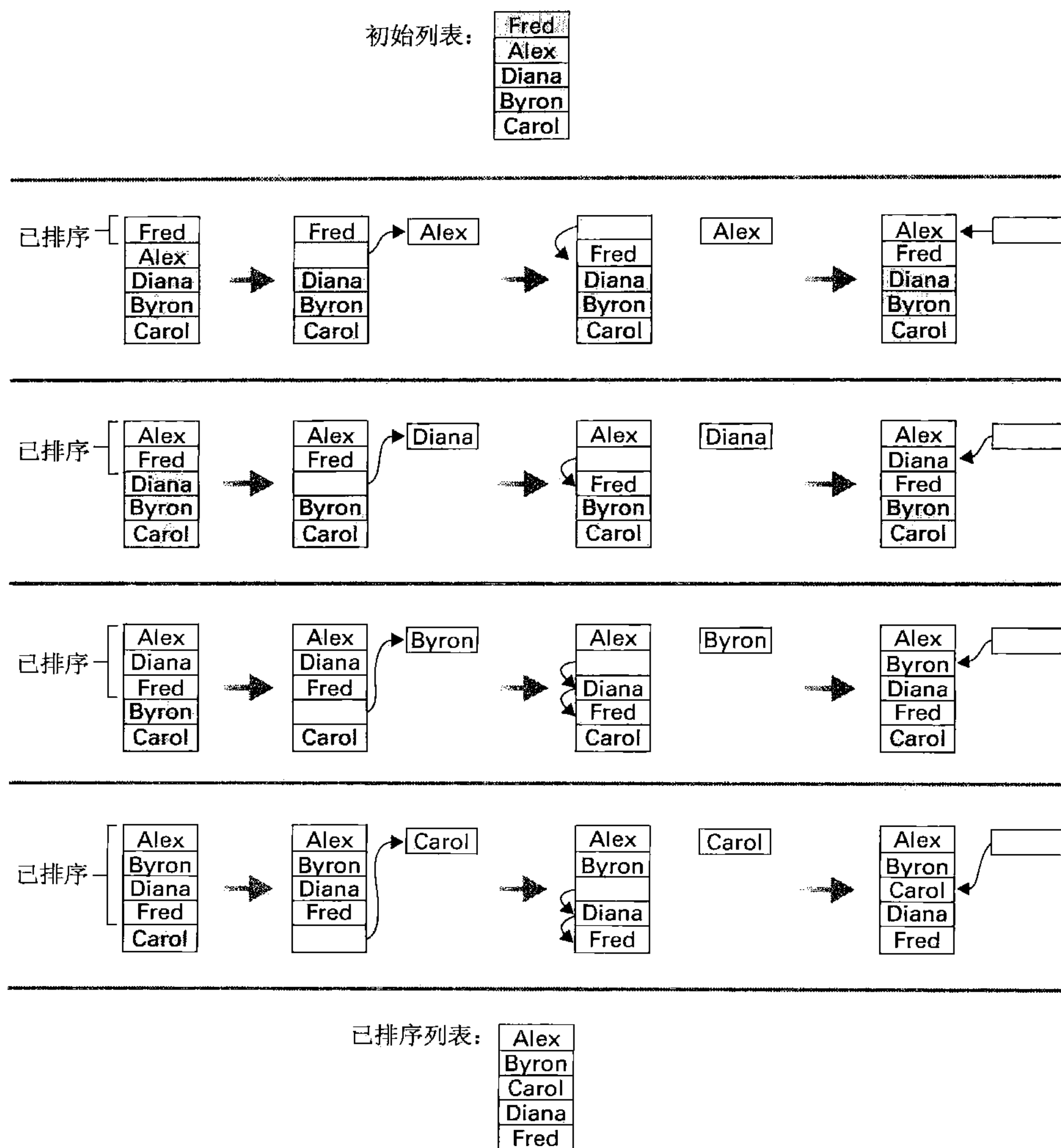


图5-10　按字母顺序为Fred、Alex、Diana、Byron和Carol排序

在分析了一个特殊列表的排序过程之后，现在的任务是将这个过程通用化，以获得对一般列表排序的算法。为了达到这个目的，我们考查图5-10中的每一行，发现它们都执行同样的通用的过程：取出列表中还未排序的部分中的第一个名字，将已经排序的部分中大于此名字的表项向后移，然后将这个取出的名插入到这一部分的空缺处。如果把取出的名字称为**主元**（pivot），那么这个过程可以用下面的伪代码表示：

```
把主元表项移到一个临时位置使该列表留出一个空位置；
while （如果这个空位置上面存在一个名字并且那个名字比主元大） do
      （把这个名字向下移到空位置上使该名字上面留出一个空位置）
把主元项插到列表的空位置上
```

下一步，我们来观察这个过程怎样被重复执行。为了开始排序过程，主元项应该是列表的第二项。然后每当再执行一次前，主元项的选择应该是从列表中的下个位置直到列表的最后一个位置。也就是说，随着前面的程序的重复，主元项的位置从第二项移进到第三项，然后到第四项，依次类推，直到程序定位到表的最后一项。我们使用下面的语句对这个过程进行控制：

```
N ← 2;
while ( N的值不超过列表的长度 ) do
  ( 把列表的第N项作为主元项;
    ⋮
 (N ← N+1)
```

这里，N 表示主元项的位置，列表的长度是指列表中的项个数，而省略号则是指放置前面例程的位置。

完整的伪代码程序如图 5-11 所示。简言之，这个程序通过重复地移动表项并且将其插入适当的位置来完成排序。由于这种重复的插入过程，这种算法称为**插入排序**（insertion sort）。

```
procedure Sort(List)
N ← 2;
while ( N 的值不超过列表的长度 ) do
    ( 把列表的第 N 项作为主元项;
    把主元项移到一个临时位置使该列表留出一个空位置;
    while ( 如果这个空位置上面存在一个名字并且那个名字比主元大 ) do
        ( 把这个名字向下移到空位置上使该名字上面留出一个空位置 )
    把主元项插到列表的空位置上;
    N ← N+1
    )
```

图5-11 用伪代码表达的插入排序算法

注意，图 5-11 中所示的结构是一个循环内部还有循环的结构，外层循环用第一个 while 语句表述，而内层循环用第二个 while 语句表述。每次执行外层循环体都导致内层循环体被初始化而且重复执行直到终止条件出现，因此外层循环体的一次执行将导致内层循环体多次执行。

外层循环控制的初始化部分通过使用赋值语句 N←2;实现。修改部分通过每次给 N 加 1（语句 N←N+1）完成。终止条件当 N 的值超过列表的长度时出现。

内层循环控制通过移动主元项并且创建一个空缺位置来初始化。循环的修改步骤通过移动表项到空位置来实现，因此导致空位置上移。终止条件在空位置上方紧临的名字不大于主元或空位置到达列表顶部的时候出现。

问题与练习

1. 修改图5-6中的顺序搜索过程，使其可用于未排序的列表。
2. 把下面的伪代码转换为等价的用repeat语句表述的程序。

```
Z← 0;
X← 1;
while (X<6)do
  (Z← Z +X;
   X←X +1)
```

3. 当今一些流行的程序设计语言使用语法：

```
While (...) do (...)
```

来表示前测试循环，而用语法：

do (…) while (…)

表示后测试循环。尽管设计上显得很优雅，但是这种相似的形式会导致什么问题?

4. 假设图5-11所示的插入排序算法用来对表Gene、Cheryl、Alice和Brenda进行排序。描述外层while结构的循环体的每次执行结束时表的构成。
5. 为什么不能把图5-11中while语句内的“大于”改为“大于等于”？
6. 插入排序的一个变种是**选择排序**（selection sort）。此排序法从选择表中最小的项并将其移至表的最前面开始，然后选择表中的剩余项中最小的项，同时将其放到表的第二个位置。通过重复从列表的剩余部分选择并前移最小的项，经过排序的部分便从前向后逐渐变长，而后面未排序的部分逐渐缩短。使用我们的伪代码来表达用选择排序法实现的类似于图5-11中的列表排序过程。
7. 另一个著名的排序算法是**冒泡排序**（bubble sort）。这个算法基于这样一种机制，重复对列表中相邻的两项进行比较，如果它们并不是依照规定排序的就交换它们的位置。我们假设等待排序的列表有n项。冒泡排序法将从比较（并可能互换位置）第n项和第$n-1$项开始。然后，它将考虑第$n-1$项和第$n-2$项之间的关系，并且继续向列表的前面移动，直到列表的第一项和第二项进行比较（并可能互换位置)。可以看出，通过一遍排序，最小的表项将被移到列表的最前面。同理，再一次排序将把仅大于最小表项的项放到列表的第二个位置。因此，重复$n-1$遍后就完成了整个列表的排序。（如果我们观察算法的工作过程，那么就会看到小表项像气泡一样冒到了列表的前面。）请使用伪代码来表达用冒泡排序法实现的类似于图5-11中的列表排序过程。

5.5 递归结构

递归结构提供了除循环模型以外用来实现重复活动的另外一种选择。循环涉及重复一个指令集，其方式是执行完成一组指令，然后重复执行，而递归则是通过将指令集作为自身的一个子任务重复调用来运行的。在处理来电的过程中，呼叫等待的特性就是一个很好的递归例子。在这个例子中，当处理另外一个来电的时候，先前未完成的通话将被搁置一边，结果是一共进行了两次通话。然而，这两次通话并不是以那种先执行一个然后再执行一个的类似于循环结构的方式完成的，而是一次通话在另外一次通话过程中进行。

5.5.1 二分搜索算法

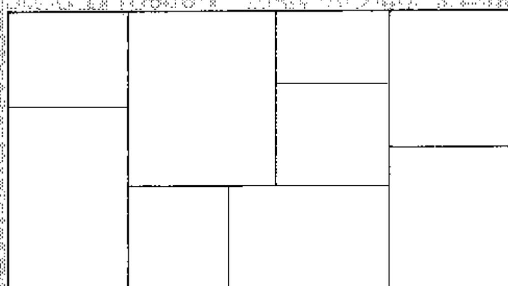
作为一种介绍递归的方法，让我们再次处理在一个已经排序的列表中查找是否存在某特定项的问题，但是这次将采用查字典时所使用的方法来考虑这个问题。在这个例子中，我们不再按照一项一项或者是一页一页的顺序进行，而是通过直接翻到我们认为目标可能存在的那一页开始工作，如果足够幸运将可能一下就找到目标，否则就必须继续查找。但至少我们已经大大缩小了查找的范围。

当然，在查字典的时候，我们有知道可能在哪能查到单词的先验知识。例如，查 somnambulism 这个单词，我们就会从字典的后面部分开始查找。但是对于一般列表，我们并没有这种先验知识，所以我们总是假定从列表的“中间”项开始查询。这里“中间”这个词之所以被引号括起来，是因为一个列表可能包含偶数个项，那么此时就没有准确意义上的中间项了。在这种情况下，我们将假定该“中间”项为该列表中后半部分的第一项。

如果列表的中间项就是要找的项，那么此时查找成功。否则，我们至少可以将查找限定在列表的前半部或者后半部，具体要依赖于所查找的目标值是小于还是大于我们的中间项。（记住，前提是所查找的列表是已经排好序的。）

艺术中的递归结构

下面的递归过程可用于在一块长方形画布上生成荷兰画家皮耶·蒙德里安（Piet Mondrian，1872—1944）风格的图画。他通过在一个长方形画布上将长方形连续划分成更小的长方形来绘画。你可以尝试自己实现这一过程来画出与下面类似的图案。从把该过程应用到你所工作的代表画布的一个长方形上开始吧。（如果怀疑由该过程表示的算法是否是一种符合5.1节定义的算法，那么你的怀疑是有根据的。实际上，这是一个非确定性算法的例子，因为有很多地方需要实现该过程的人或机器来作出“创造性”的决定。也许这就是为什么蒙德里安的结果被认为是艺术，而我们的却不是。）



```
procedure  Mondrian(Rectangle )
if ( 用你的艺术眼光看长方形太大 )
  then ( 把长方形划分成2个较小的长方形;
         把Mondrian过程应用到一个较小的长方形;
         把Mondrian过程应用到另一个较小的长方形 )
```

查找和排序

顺序搜索算法和二分搜索算法仅是许多实现搜索过程的算法中的两个。同样，插入排序是许多排序算法中的一个。其他经典的排序算法包括归并排序（见第12章）、选择排序（见5.4节问题与练习6）、冒泡排序（见5.4节问题与练习7）、快速排序（对排序过程采取分治的方法）和堆排序（使用一种巧妙的技术，可以在列表里找到应该向前移的表项）。有关这些算法的讨论可以参阅本章末的“课外阅读”里面列出的书目。

在列表的剩余部分查找，我们可以使用顺序搜索法，但这里仍然使用在完整列表中所使用的方法在表的剩余部分进行查找。也就是说，我们选择列表的剩余部分的中间项作为下一个要考虑的项。像刚才那样，如果这个项就是我们要找的，那么查找结束，否则将把查找的范围限定在更小的区域中。

图 5-12 简要概括了这个查找方法。这里我们要查找的是图左边列表中的 John 表项。首先考虑中间项 Harry。可以看出，所查找的目标属于后半部分，接下来的查找将在原始表的后半部分开始。该子表的中间项是 Larry，显然所查找的目标在 Larry 之前，所以我们的注意力将转到当前子表的前半部分。查看第二个子表的中间项时，我们找到了目标表项 John，此时查找成功。简言之，我们的策略是将被讨论的列表连续地分成更小的段，直到最终找到目标或者发现查找被限制在一个空段中。

这里需要强调最后一点。如果所查找的目标值不在原始表中，那么我们的查找方法会将列表不断分成更小的段直到所考虑的段为空，此时算法认为查找失败。

图 5-13 就是整个算法的一个伪代码草稿。这个草稿引导我们通过测试表是否为空来开始查找过程。如果表为空，我们会被告知查找失败。否则，我们被告知要考虑中间项。如果此项不是目标值，我们就被告知要查找表的前半部分或后半部分。这两种可能都需要第二次查找。显然，如

果通过调用一个抽象工具的服务来执行这种查找将十分方便。尤其是，当我们应用一个称作Search的过程来执行第二次查找时。因此，为了完成我们的程序，就必须提供这样一个过程。

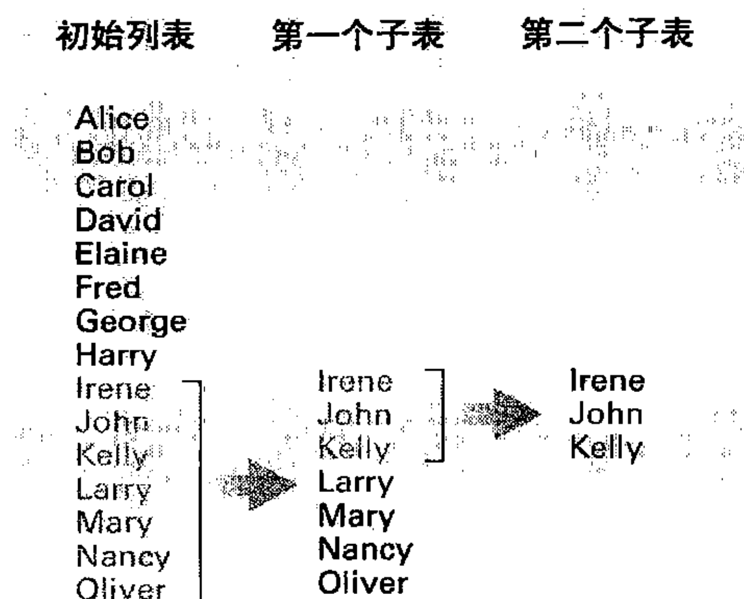


图5-12 使用我们的策略在列表中查找John表项

```
If ( 表为空 )
 then
  ( 报告查找失败 )
 else
  [选择List的中间项作为 TestEntry;
   执行以下与条件相符的case指令块
     case 1: TargetValue = TestEntry
          ( 报告查找成功 )
     case 2: TargetValue < TestEntry
          ( 在List中位于TestEntry项之前的部分查找 TargetValue,并报告查找结果 )
     case 3: TargetValue > TestEntry
          ( 在List中位于TestEntry项之后的部分查找 TargetValue,并报告查找结果 )
]end if
```

图5-13 二分搜索技术的草稿

但是这个过程应该执行相同的任务，而这个任务已经由前面所给出的伪代码表达。第一步，我们将检查给定表是否为空，如果非空，它将开始考虑此表的中间项。因此我们可以提供一个过程，只需把当前例程视为Search过程，并在二次查找的地方插入对这个过程的另外一次引用。结果如图5-14所示。

注意，这个过程包含了一个对自身的引用。当然，如果遵循这个过程，然后到达指令

应用Search过程……

我们将把同样的过程应用到一个较小的表上，而这正是应用于原始列表的那一过程。如果首次查找成功，我们将返回并声明初始查找成功；如果第二次查找失败，我们将声明初始查找失败。

为了理解图5-14中的过程是如何运行的，我们假定一个表包括Alice、Bill、Carol、David、Evelyn、Fred和George，查找的目标值是Bill。首先选择David（中间项）作为考虑的测试项。因为目标值（Bill）小于该测试项，我们将对David之前的列表项调用Search过程（此时该列表是Alice、Bill

和 Carol)。这样我们便创建了 Search 过程的第二个副本，并将它用于第二次查找任务。

```
Procedure Search ( List, TargetValue )
if ( 表为空 )
  then
      ( 报告查找失败 )
  else
      [选择List的中间项作为TestEntry;
       执行以下与条件相符的case指令块
        case 1: TargetValue = TestEntry
              ( 报告查找成功 )
        case 2: TargetValue < TestEntry
             ( 应用Search过程查看TargetValue是否在List中位于TestEntry
               项之前的部分，并报告查找结果 )
        case 3: TargetValue > TestEntry
             ( 应用Search过程查看TargetValue是否在List中位于TestEntry
               项之后的部分，并报告查找结果 )
]end if
```

图5-14 二分搜索算法的伪代码

现在我们拥有两个正在执行的查找过程，如图 5-15 所示。先前的原始副本在执行

应用Search过程查看TargetValue是否在List中位于TestEntry项之前的部分

指令时被暂时挂起，此时我们应用第二个副本完成对列表 Alice、Bill 和 Carol 的查找任务。完成第二次查找后，我们将放弃该过程的第二个副本，并将它找到的结果通知原始副本，然后继续原始的过程。通过这种方式，过程的第二个副本被当作原始过程的子过程运行，完成由原始的模块请求的任务，然后消失。

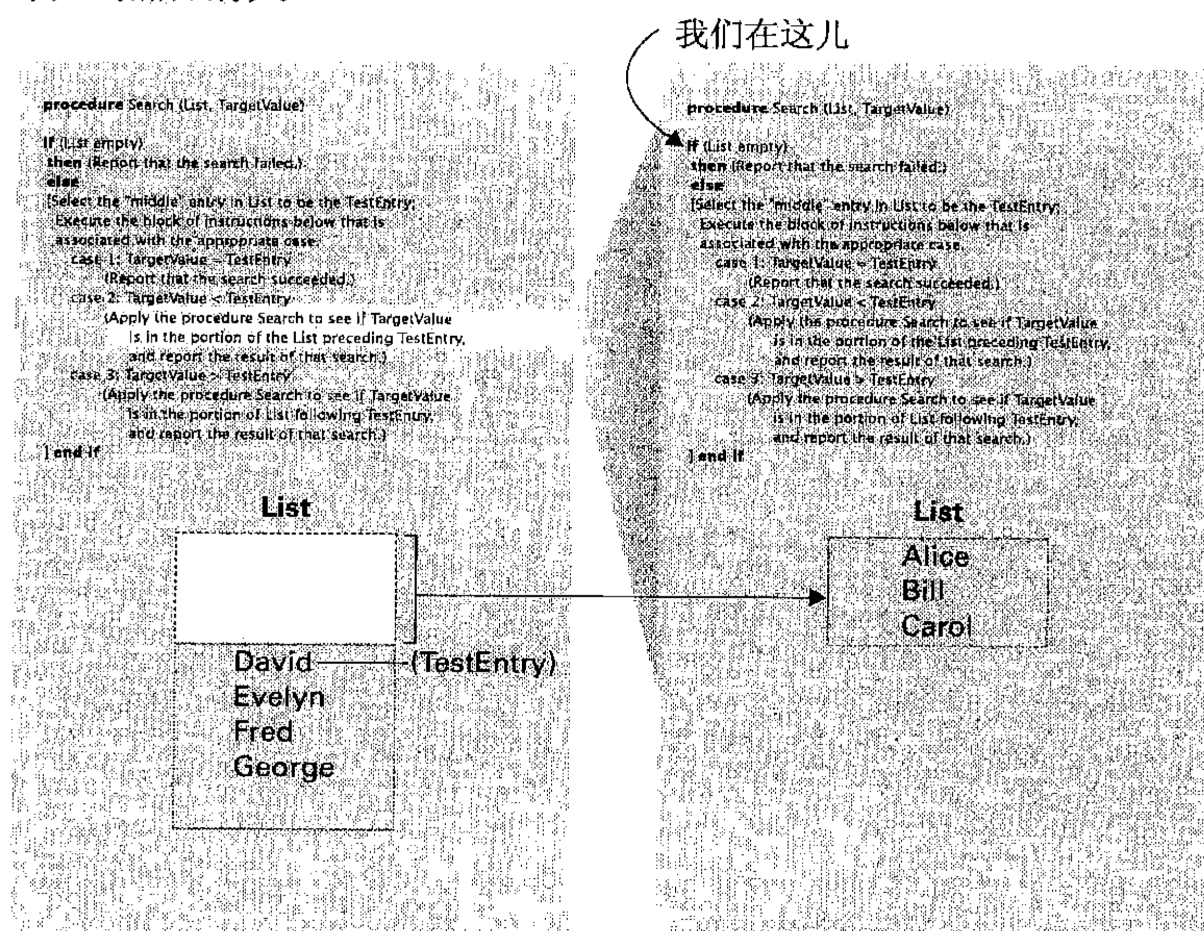


图 5-15

第二次查找中，Bill 被选作测试项，这是因为它是表 Alice、Bill 和 Carol 的中间项。由于这一项与所查找的目标相同，于是此过程就宣告整个查找成功并结束。

此时，我们已经把过程原始副本所请求的第二次查找完成，所以可以继续原始副本的执行。此时我们又被告知需要把第二次查找的结果作为原始查找的结果报告。因此，最终得到结论，即原始查找成功。我们的查找过程正确结束，结果表明 Bill 是表 Alice、Bill、Carol、David、Evelyn、Fred 和 George 中的成员。

现在让我们考虑，如果要求图 5-14 所示的过程在表 Alice、Carol、Evelyn、Fred 和 George 中查找 David，将会出现什么情况呢？这次过程的原始副本选择 Evelyn 为测试项，并推断目标肯定存在于表的前半部分。因此它请求过程的另外一个副本对 Evelyn 之前的表进行查找——也就是包含 Alice 和 Carol 两项的表。在此阶段，我们遇到了图 5-16 所示的情况。

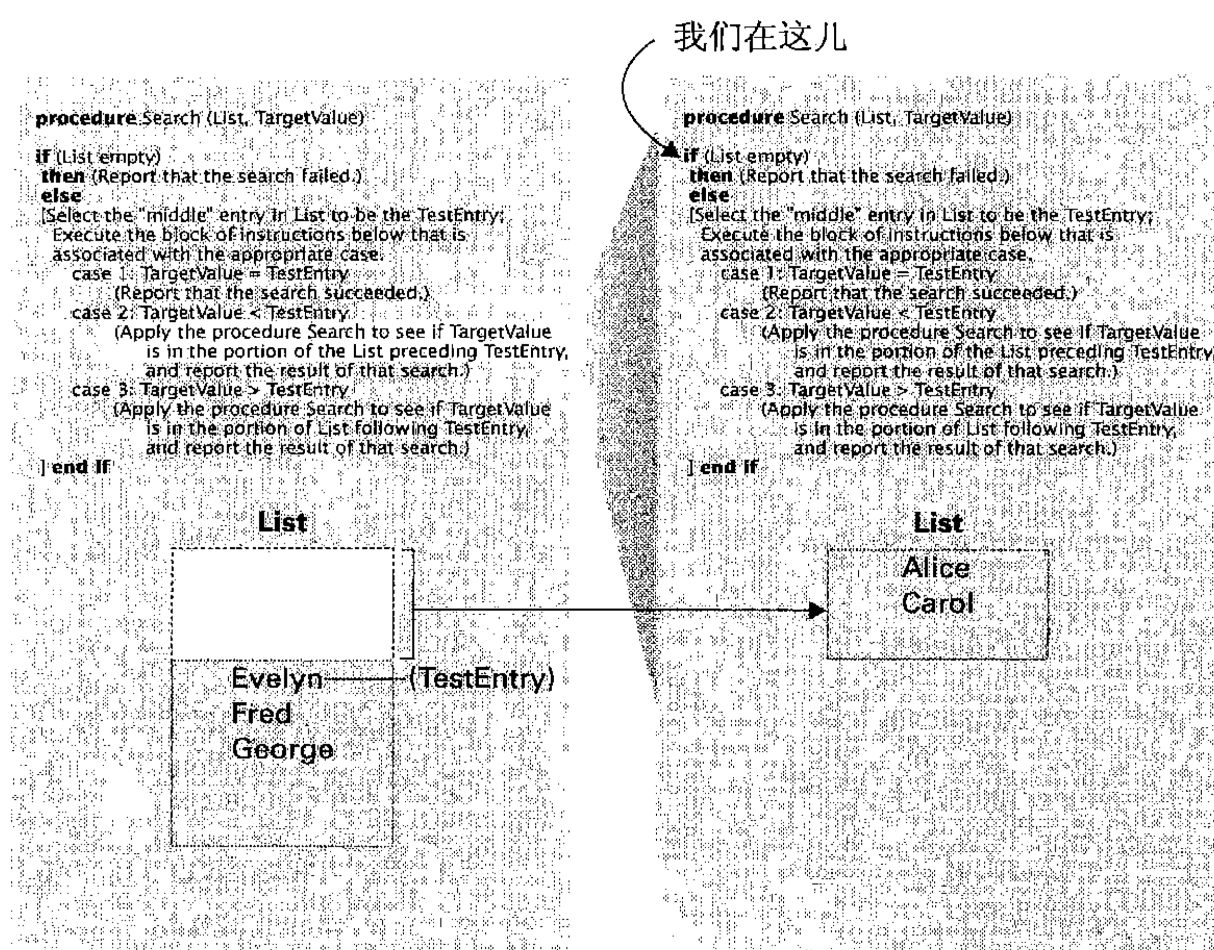


图　5-16

过程的第二个副本选择 Carol 当作当前项，同时推断目标必定存在于该表的后半部分。然后，它将请求过程的第三个副本来寻找 Alice 和 Carol 组成的表中 Carol 后面的名字组成的表。该子表是空表，所以过程的第三个副本需要在一个空表中寻找目标项。图 5-17 显示了目前所处的情况。过程的原始副本处理表 Alice、Carol、Evelyn、Fred 和 George 的查找任务，测试项是 Evelyn；第二个副本处理表 Alice 和 Carol 的查找，测试项是 Carol；第三个副本将要在一个空表中开始查找。

当然，过程的第三个副本很快就会推断它的搜索已经失败并终止运行。第三个副本的任务的完成使得第二个副本能够继续执行。第二个副本发现它请求的查找失败，于是宣布自己的查找失败并且终止。原始副本等到了第二个副本的报告，因此继续执行。过程的原始副本得到这个消息后推断自己的过程失败并终止。我们的例程正确地推断 David 不在 Alice、Carol、Evelyn、Fred 和 George 组成的表中。

综上所述，如果回顾前面的例子，我们能够看到图 5-14 所示的算法重复地将所考虑的列表分成两个较小的块，并将后续的查找严格限制在其中一个块中。这种一分为二的方法就是该算法称为**二分搜索**（binary search）的原因。

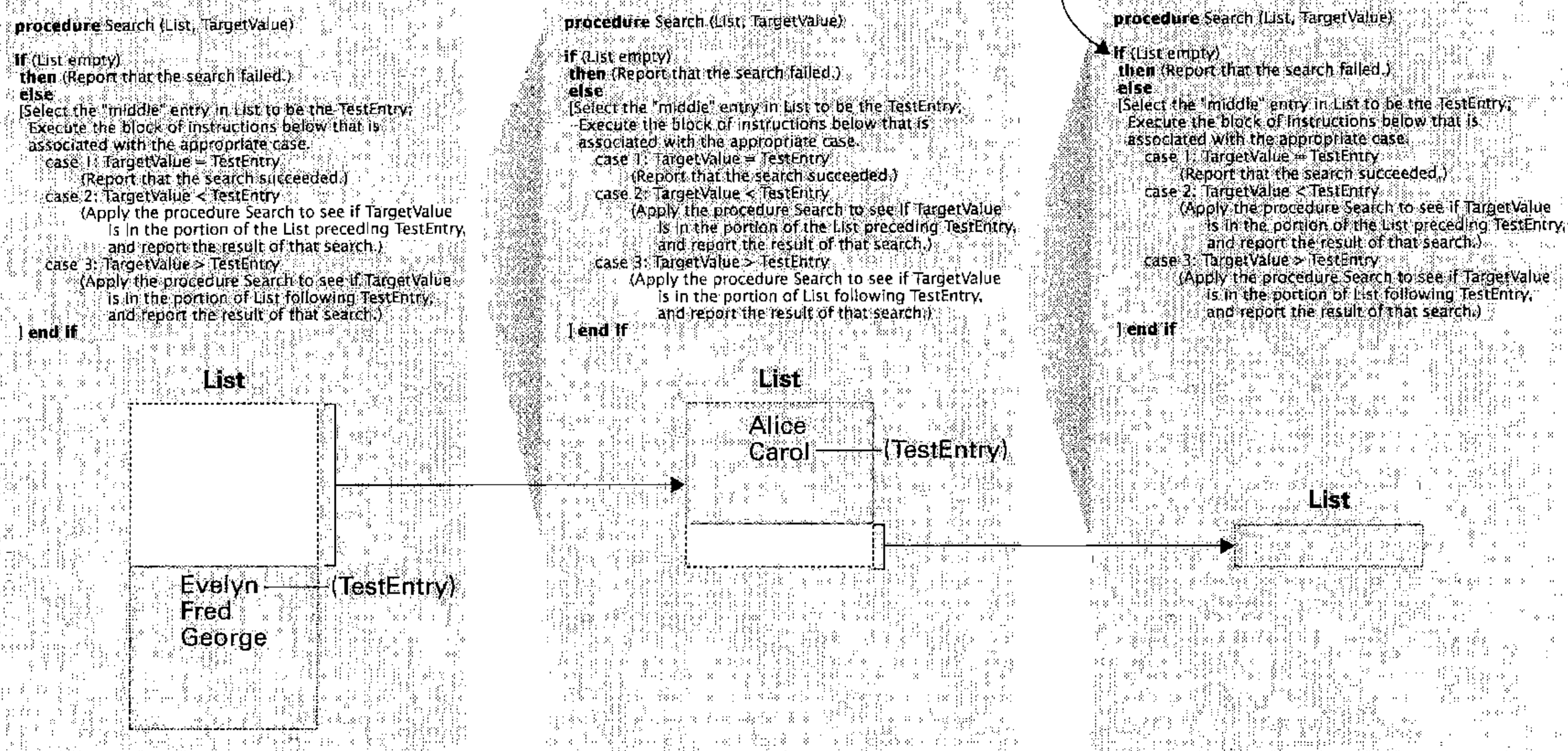


图 5-17

5.5.2 递归控制

二分搜索算法与顺序搜索两者的相似之处在于都需要执行一个重复过程。但是，这种重复的实现却是截然不同的。顺序搜索是以一种循环的方式重复执行一个过程，而二分搜索则是把每一阶段重复当做前一阶段的子任务。该技术称作**递归**（recursion）。

正如我们所见的，递归过程的执行之所以容易使人困惑在于它对于过程副本的多重调用，每一次调用都称为一次过程的激活。这些激活通过一种嵌套方式动态创建，并随着算法的前进而最终消失。在任何给定的时间中所有存在的激活，只有一个是正在执行的，其他的都处于等待状态，等待另外的激活终止后方可继续。

作为一个重复的过程，递归系统也依赖于与循环结构相似的正确控制方式。就像循环控制一样，递归系统也依赖于对终止条件的测试，同时也必须保证终止条件能够达成。事实上，正确的递归控制应该包含与循环控制中相同的 3 个组成部分——初始化、修改和终止测试。

通常，递归程序将终止条件［通常称作**基本条件**（base case）或者**退化条件**（degenerative case）］的测试设计在请求继续激活之前，如果不满足终止条件，例程就创建自己的另外一个激活状态并且分配这个状态解决一个距离终止条件更近的修订问题的任务。但是，如果满足终止条件，现有的激活状态就会终止，并不再创建任何其他激活状态了。

让我们来看看图 5-14 中的二分搜索过程是如何实现重复控制的初始化和修改阶段的。在这个例子中，一旦目标值被找到或者任务被缩减到只是完成对空表的查找，额外激活的创建就会停止。整个过程是从简单地给出初始表和目标值开始的。从该初始配置开始，过程就会将其任务修改为在更小的表中进行查找。因为原始表的长度有限，并且每次修改步骤都会减小所考虑的表长度，所以我们可以确保目标值最终会被找到或者任务最终会被缩减到对空表进行查找。因此，我们能够推断这个重复过程一定可以结束。

最后，由于循环和递归控制结构都是用来完成一系列指令重复运行的方法，我们可能会问这两种结构是否在能力上等价。换句话说，如果一个算法被设计成循环结构，那么是否存在一个递归结构算法，这个算法也可以完成前面那个循环结构算法所要解决的问题，或反之？这样

的问题在计算机科学中是非常重要的，因为其答案告诉我们什么特性应该被提供给程序设计语言，以便获得可能的最强大的程序设计系统。我们会在第 12 章中回到这个问题，那里将考虑更多计算机科学理论方面的问题以及它的数学基础。带着这个背景，我们可以证明附录 E 中提到的“迭代和递归结构的等价性”。

问题与练习

1. 在名单Alice、Brenda、Carol、Duane、Evelyn、Fred、George、Henry、Irene、Joe、Karl、Larry、Mary、Nancy和Oliver中按照二分搜索（图5-14）查找Joe的时候，在查找过程中会检查哪些名字？
2. 使用二分搜索在200项的表中进行查找时所需查找的最大项数是多少？如果是100 000项的表又会如何呢？
3. 如果N的初始值为1，那么以下递归过程会打印出什么样的数列？

```
procedure Exercise(N)
打印 N 的值;
if(N<3) then (对 N+1 应用 Exercise 过程);
打印 N 的值;
```

4. 问题与练习3中的递归过程的终止条件是什么？

5.6 有效性和正确性

在本节中，我们介绍两个构成计算机科学中重要的研究领域的主题。第一个是算法有效性，第二个是算法正确性。

5.6.1 算法有效性

尽管当今的计算机每秒可以处理数百万条指令，有效性仍旧是算法设计中所关注的一个主要问题。通常，在效率高低的两个算法之间的选择能够产生对于问题的实用或者不实用的两种解。

让我们考虑这样一个问题，一个大学的教务主任需要面对检索和更新学生记录的任务。尽管在任何一个学期学校可能只有大约一万名学生注册，但是它的“当前学生”文件包括了超过三万条的学生记录，这些学生在过去的几年中至少注册了一门课程但并没有完成学业，所以在某种意义上被认为是现在的学生。眼下，让我们假设这些学生的记录以表的形式存储于教务主任的计算机中，这些表依照学号顺序排列。为了寻找任何一条学生记录，教务主任要在这张表中查找特定的学号。

我们已经讲述了两种可用于在这些已排序的表中进行查找的算法：顺序搜索法和二分搜索法。现在的问题是，对于教务主任，这两种算法是否会带来不同的效果。我们首先考虑顺序搜索法。

给定一个学生的学号，顺序搜索算法从表的开头开始，将所有的记录与期望学号相比较。因为不知道关于目标值的任何原始信息，我们不能推断究竟要查找多少条记录才能得到结果。但是，在多次查找之后，我们认为平均查找深度是表的一半长度；有的可能短一些，有的可能长一些。因此经过一段时间我们估计，顺序搜索平均每次大概需要检查 15 000 条记录。如果检索并且检查每一条记录需要 10 ms，那么这样的查找平均需要 150 s——这个时间对于等待计算机显示学生记录的教务主任来说是不可忍受的。即使检索和检查每条记录只需要 1 ms，那么整

个查找仍然需要大概 15 s，这个等待时间仍然较长。

相反，二分搜索算法通过比较目标值和表的中间项来进行查找。如果中间项不是期望的记录，则至少将查找限制在原始表的一半。因此，在检查一个有 30 000 条记录的表的中间项之后，二分搜索需要再次考虑的记录数量最多 15 000 条。在第二次搜索之后，最多还剩余 7500 条，然后在第三次后，又会降至不多于 3750 条记录，照此继续，我们发现最多 15 次之后，目标值就应该在这个有 30 000 条记录的表中被找到。因此，如果每一次记录检索需要 10 ms，那么查找一个特定记录的过程就只需要 0.15 s——这意味着，对于教务主任来说访问任何特定记录可以瞬间完成。我们得出结论：在顺序搜索算法和二分搜索算法之间的选择将对该应用产生巨大影响。

这个例子表明了计算机科学领域中大家熟知的算法分析的重要性，这种分析包含了对于资源的研究，比如算法需要消耗的时间或者存储空间资源。这种研究的一个主要应用在于给出了对于二选一算法之间不同优点的评估。

算法分析通常包括最优情况分析、最差情况分析和平均情况分析。在上面的例子中，我们通过分析平均情况下的顺序搜索法和最差情况下的二分搜索算法，估计了在30 000条记录中完成查找所需的时间。通常这种分析应该在更为普通的情况下进行。也就是说，当考虑查找算法的时候，我们不能关注于表的特定长度，而是要尝试列出某种可以在表示任何长度的表中进行查找的算法的性能公式。基于我们之前的推理，不难得出对于任意长度的表均有意义的公式。具体而言，当需要在长度为n的表中应用时，顺序搜索算法的平均查找长度是$n/2$，而二分搜索算法在最差情况下的查找长度不超过$\lg n$。（$\lg n$表示以2为底n的对数。）

我们现在用类似的办法分析插入排序算法（如图 5-11 所示）。回想这个算法涉及选择表项（该项称为主元项），将此项与其之前的那些项比较直到找到正确的插入位置，然后将主元项插入这个位置。因为该算法主要涉及两个项之间的比较，我们的方法是计算表长度为 n 时这种比较的次数。

算法从把表的第二项当做主元开始。然后，算法继续选择后面的表项作为主元直到表的末尾。在最佳情况下，每一个主元都已经在合适的位置，因此它只需要与一项进行比较。因此，最佳情况下应用插入排序到一个有 n 项的表并排序需要进行 $n-1$ 次比较（第二项与一项比较，第三项与一项比较，依次类推）。

相反，在最差情况下，每一个主元都必须与表中排在它前向的所有记录进行比较，然后才能找到合适的位置。这种情况发生在表被反向排序的时候。在这种情况下，第一个主元（表的第二条记录）要与一项进行比较，第二个主元（表的第三条记录）要与两项进行比较，依次类推（如图 5-18 所示）。因此，在给长度为 n 的表排序时总共需要进行的比较次数为 $1+2+3+\cdots+(n-1)$，也就是 $\frac{1}{2}(n^2-n)$。具体而言，如果一个表包含 10 项，在最差情况下插入排序法需要进行 45 次比较。



图5-18 将插入排序应用于最差情况中

在插入排序平均情况中，我们期望每一个主元与表中在它前面项的一半进行比较。这样一来，

一共需要执行的次数是最坏情况的一半，也就是 $\frac{1}{4}(n^2-n)$ 次比较可以完成对 n 项表的排序。例如，假使用插入排序算法为长度为 10 的多个表排序，平均情况下每次排序需要 22.5 次比较。

这些结果的重要性在于插入排序法执行中的比较次数给出了执行这种算法对时间的大概需求量。使用这个估算，图 5-19 显示了一个当列表的长度增加时，执行插入排序算法所需时间如何增长的示意图。该图是基于我们对于算法最坏情况的分析。在此分析中，我们推算出在长度为 n 的列表中进行排序最多需要 $\frac{1}{2}(n^2-n)$ 次比较。在图中，我们标出了几个表的长度，同时给出了在每一种情况下需要的时间。注意，当列表的长度等步长增加时（即每次增加的长度相同），排序需要的时间的增长速度更快。因此，这个算法在列表长度增加的时候，效率会变得越来越差。

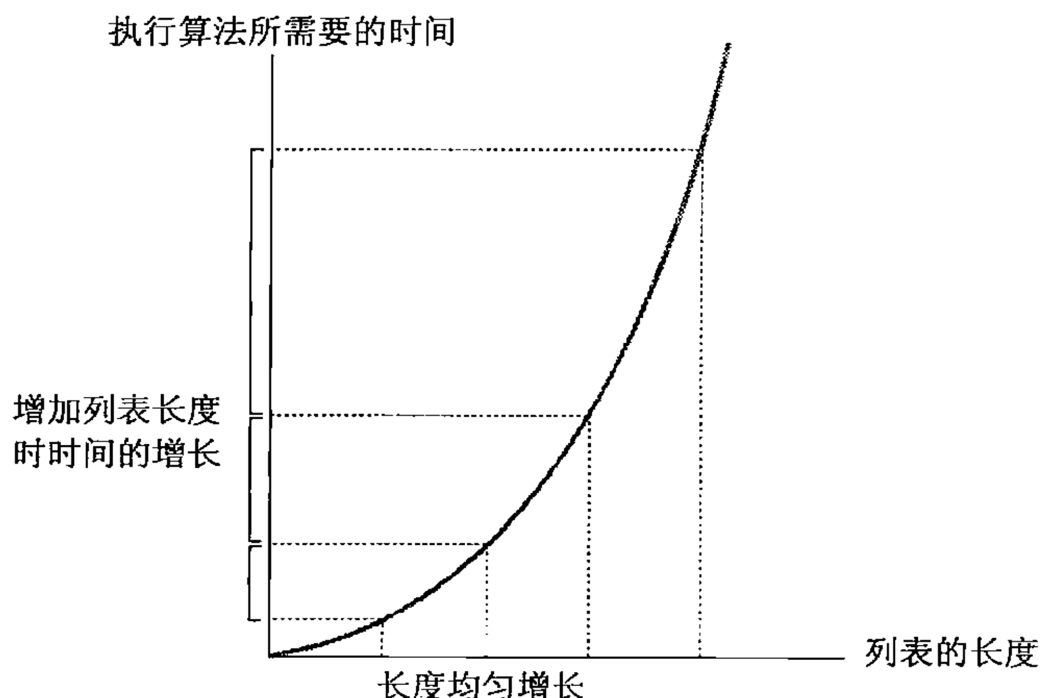


图5-19　插入排序算法的最差情况分析图

让我们使用相似的方法来分析一下二分搜索算法。回想前面我们推导得到使用该算法在长度为n的列表中进行查找的时候需要最多查询$\lg n$项，这可以帮助估算对不同长度的列表执行这一算法所需要的时间。图5-20给出了基于这种分析的一张曲线图，我们依旧标出几个等步长增加的列表的长度和每种情况下算法执行所需要的时间。注意，算法随着列表长度的增加，对于时间需求的增加是在逐步递减的。也就是说，二分搜索法在较长的列表中效率更高。

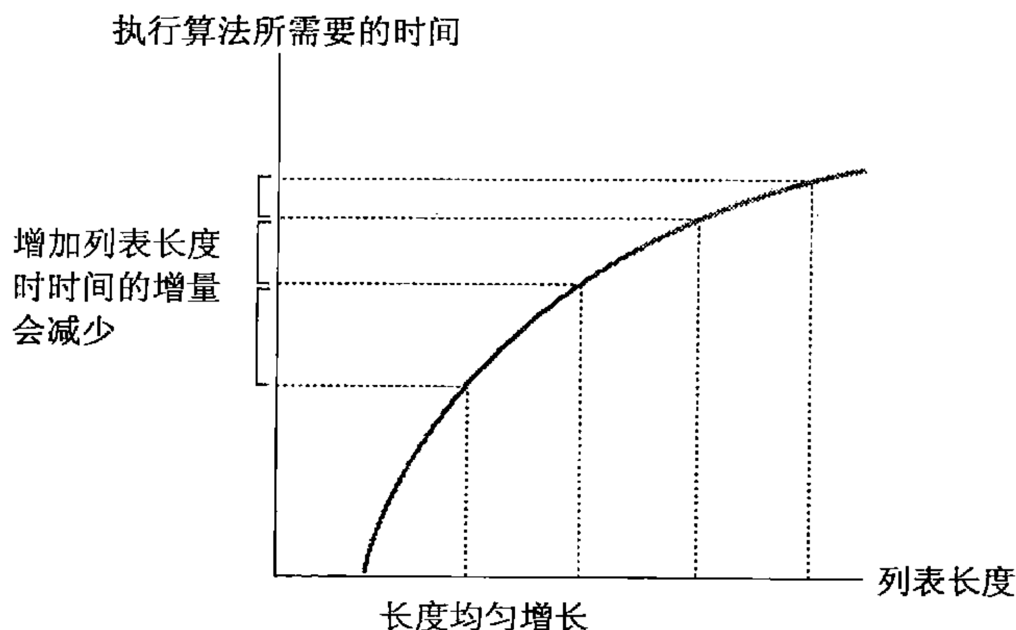


图5-20　二分搜索法的最差情况分析图

图 5-19 和图 5-20 的区别在于图的大体形状，该大体形状揭示了一个算法在应对越来越大的输入规模的情况下到底有多好。此外，一个图像的大体形状是由表达式的类型而非具体的表达式所确定的——所有的线性表达式的图像都是一条直线，而二次表达式则给出一条二次曲线，所有的对数表达式都可以得到图 5-20 中的对数曲线。习惯上我们用可以产生一个形状的最简单表达式来标识该形状，具体而言，我们用表达式 n^2 来识别二次曲线，而用 lg$n$ 来表示对数曲线。

我们已经看到了，通过比较执行一个算法需要的时间与其输入娄据大小所得的图形可以反映该算法的效率特性，因此可以根据这些图的形状对算法进行分类——通常是基于算法的最差情况分析。用来区分这些类型的符号有时称作**大Θ标记**。所有图为二次曲线的算法，例如插入排序法，都被划归为$\Theta(n^2)$表示的算法类型；所有图像为对数曲线的算法，比如二分搜索法，都被分到$\Theta(\lg n)$表示的算法类型中。知道特定算法属于的类型使我们可以预测它的性能，并且可以拿它与能够完成相同工作的算法进行比较。两个$\Theta(n^2)$算法在输入大小增加的时候，对于时间将会有相近的需求变化。此外，一个$\Theta(\lg n)$算法就不会像$\Theta(n^2)$算法那样随着输入大小的增加对时间的需求扩张得如此剧烈。

5.6.2 软件验证

回想波利亚对于问题求解的分析（5.3 节），其中第 4 阶段就是对问题解决方案的准确性和其作为求解其他问题的工具的潜力进行评价。这个阶段第一部分的重要性由下面的例子体现出来：

> 一位拿着由 7 个金环组成的链子的旅行者必须在一个饭店里住 7 夜。每一夜的租金是金链中的一环。应该怎样对链子进行最少次数的切割，旅行者才能每天早上支付旅店的一环而不用提前支付住宿费？

首先我们认识到并不是每一环都必须被切开。如果只切开第二个环，那么我们就可以让第一个环和第二个环与另外 5 个环分开。按照这个想法，我们得到这样一种解，就是只需要切割链中的第二、第四和第六个环，这个过程将所有的环分开而只对 3 个环进行了切割（如图 5-21 所示）。此外，任何更少次数的切割都会留下两个仍然连在一起的环，所以我们推断这个问题的正确答案应该是 3 次。

进一步考虑这个问题，在只有第三个环被切开的时候，我们获得了 3 部分金链，长度分别是 1、2 和 4（如图 5-22 所示）。对这些块，我们可以进行如下操作。

第一天早上：给饭店一个环。

第二天早上：给饭店一个两个环的金链，同时找回一个环。

第三天早上：给饭店一个环。

第四天早上：把4个环的金链给饭店，同时找回原先给饭店的那3个环。

第五天早上：给饭店一个环。

第六天早上：给饭店一个两个环的金链，同时找回一个环。

第七天早上：给饭店一个环。

结果，第一个答案，也就是那个我们确定是正确的方法，实际上是错误的。但是，我们又如何认定新方法是正确的呢？可能这样反驳：因为一个环必须在第一天早上给饭店，所以至少要从金链上切一个环下来，同时因为新办法只需要一次切割，所以必定是最优的。

转换到程序设计环境中，这个例子强调了一个被认为正确的程序和一个正确的程序之间的区别，二者并不一定相同。数据处理领域充满了可怕的事情，比如尽管“知道”一个软件是正确的，但最终还是因为一些没有预料到的情况而在关键的时刻发生错误。因此软件验证很重要，并且发现有效的验证技术也成为了计算机科学中一个活跃的研究领域。

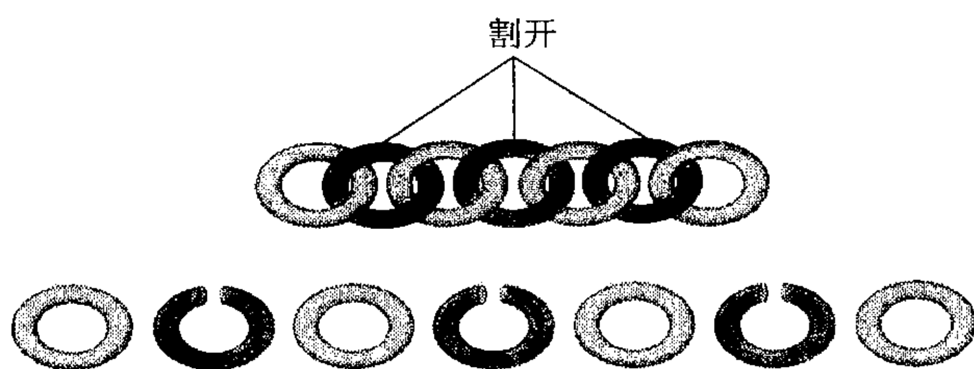


图5-21　用3次切割将链子分开

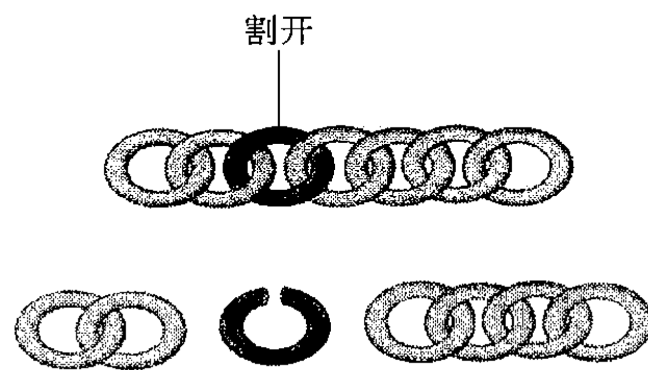


图5-22　只用1步将链子分开

在这个领域内，研究的一条主线尝试把形式逻辑技术用于证明一个程序的正确性。也就是说，目标是用形式逻辑来证明程序表达的算法确实做了它试图做的工作。基本的课题是通过将验证过程化为一个形式化过程，防止那些可能与直觉有关的不准确的推断，就像金链问题一样。让我们更详细地讨论如何把这个方法应用于程序验证中。

软件验证之外

文中所讨论的验证问题并不是软件独有的。确认执行程序的硬件没有错误是一个同等重要的问题，这涉及电路设计和机器构造的验证。而且，质量的优劣非常依赖于测试，就像在软件中所做的那样，这意味着任何一个细微的错误都可能出现在最终的产品中。一个例子是20世纪40年代由哈佛大学建造的马克一号计算机，它包含了很多布线错误，而这些错误很多年都没有被发现。一个更近的例子是在早期奔腾微处理器中浮点部分出现的错误。在这两个例子中，错误都是在产生严重后果之前被发现的。

就好像形式数学证明基于公理（几何证明通常基于欧几里得定理，然而其他证明可能基于集合论的公理），一个程序正确性的形式证明基于设计程序所使用的规格说明。为了证明一个程序可正确地为一个姓名列表进行排序，不妨假设程序的输入是一个姓名列表，如果一个程序是设计用来计算一个或者更多正数的平均值，则假设实际上输入由一个或多个正数组成。简言之，正确性证明是从对于确定条件的假设开始的，这个条件称作**前提条件**（precondition），以此来满足程序执行开始的需要。

正确性证明的下一步是考虑这些预设条件的结果是如何在程序中传播的。为了这个目的，研究人员分析了各种各样的程序结构来确定一个语句（一个在结构执行前被认为是真的语句）是如何受到结构执行的影响的。作为一个简单的例子，如果在指令 X←Y 之前，一个关于 Y 值的确定语句就已经得到，那么同样的关于 X 的语句就可以在指令执行以后获得确认。更准确地讲，如果在指令执行之前已知 Y 的值不为 0，那么也可以推断，指令执行以后 X 也一定不为 0。

一个稍微复杂的例子发生在下面这样的 if-then-else 结构中：

```
if (条件) then (指令A)
          else (指令B)
```

此处，如果某个已知的语句在结构执行以前已经获得，那么在执行指令 A 之前，我们立即知道那个语句以及测试条件皆为真，相反如果指令 B 将被执行，我们知道语句和测试条件一定为假。

依照这些规则，可以通过识别语句，也就是**断言**（assertion），来进行正确性证明，断言能够在程序的不同点建立。所得到的结果是一个断言集合，每一项都是程序预设条件的一个结果以及可以导致在程序中某点建立断言的一组指令。如果在程序结尾建立的断言可以得到相应的

输出［称为**后继条件**（postcondition）］，我们就能够断定程序是正确的。

作为一个例子，考虑图 5-23 中所示的一个典型 while 循环结构。假设，作为在 A 点的已知前提条件的一个结果，循环过程中每次终止测试的时候（B 点），我们都能确认一个特定断言为真。对于一个存在于某个循环内部的某个断言，如果在每次执行到这个循环的这一点时均为真，则称作**循环不变式**（loop invariant）。然后，如果循环一旦终止，C 点就会开始执行。此处我们可以推断循环不变式和终止条件此时均成立。（循环不变式仍旧成立是因为终止测试不改变程序中的任何值，终止条件成立是因为循环到此已经结束。）如果这此组合语句暗示着期望的后继条件，我们的正确性证明仅仅通过初始化和修改最终导致终止条件的循环组件就可以完成。

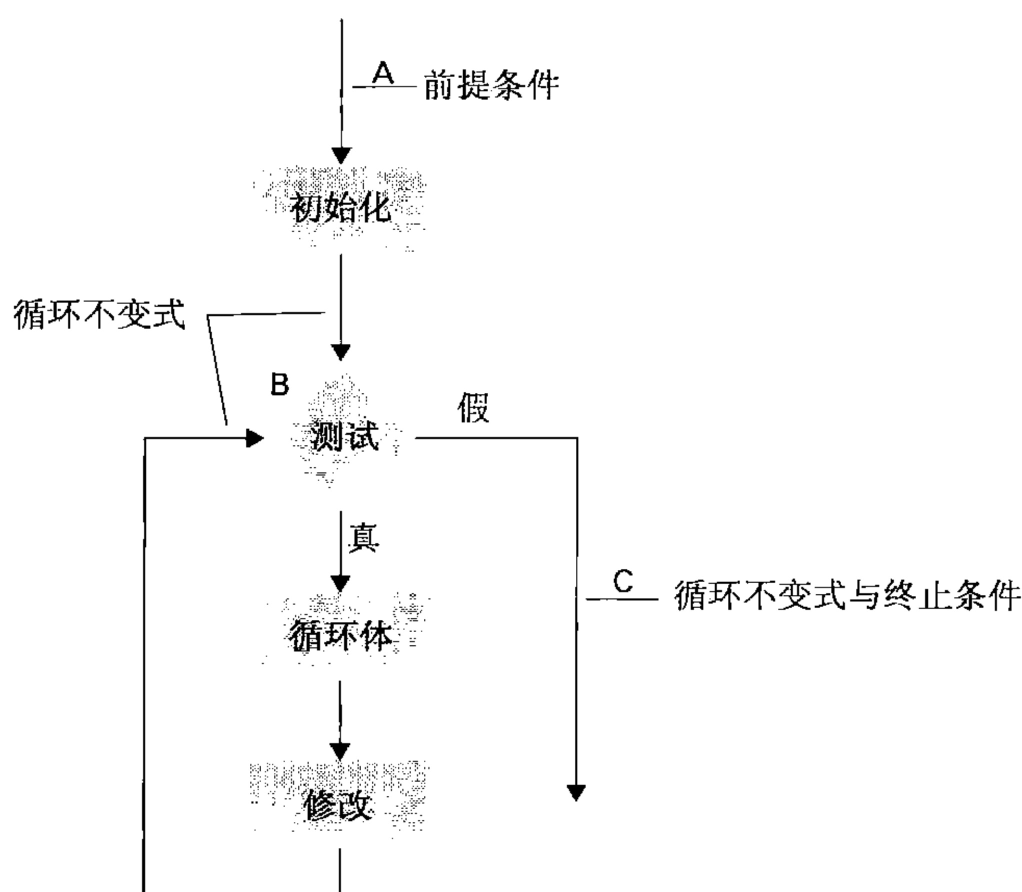


图5-23　与典型while结构相关联的断言

应该拿这个分析与图 5-11 中关于插入排序的例子相比较。那个程序的外层循环基于下面的循环不变式：

每次终止条件测试执行的时候，从位置1到位置$N-1$之间的项都完成了排序

并且终止条件是

N的值大于列表的长度。

因此，如果循环终止，我们知道两个条件均已满足，这暗示整个列表已经排序。

程序验证技术发展的进步依然非常具有挑战性。即便这样，还是取得了一些进展，其中一个更具重要性的进展是在编程语言SPARK中发现的，SPARK语言与更为流行的Ada语言之间有着紧密的联系。（关于Ada语言，我们将在下一章举例说明。）除了允许程序用像伪代码这样的高层形式表示之外，SPARK还提供给程序员包含判断的方法（如程序里的前提条件、后继条件和循环不变式）。这样，用SPARK语言编写的程序不仅仅包含了应用的算法，还包含了形式化正确性证明技术应用所需的信息。迄今为止，SPARK已经成功地应用于涉及关键软件应用的很多软件开发项目中，包括美国国家安全局的安全软件、美国洛克希德马丁公司的C130J大力神运输机的内部控制软件以及关键铁路运输控制系统。

尽管SPARK成功了，但形式化程序验证技术并没有得到广泛的应用，因此今天大多数的软件通过测试流程来进行“验证”，这个流程也还是不可靠的。毕竟通过测试进行的验证仅能说明程序对

于测试的案例是正确的，而且任何附加的结论都仅仅是推测，程序中所包含的错误往往都是测试过程中和程序开发中一些没有注意到的细微疏忽的结果。因此就像我们在黄金链中的问题一样，即使做了很大的努力去避免它，程序中的错误往往还可能不被发现。AT&T发生过一个戏剧性的例子，控制114交换站的软件中有一个错误，从1989年12月安装起到1990年1月15日都未被发现，在这段时间里，一组独特的环境致使大约500万次呼叫被不必要地阻塞。

问题与练习

1. 假设有一台使用插入排序算法编程的机器，排序100个名字的列表平均需要1 s，估算一下对1000个名字的表排序需要多长时间？对10 000个名字的表排序呢？
2. 为下面的每种类型列出一个算法的例子：$\Theta(\lg n)$、$\Theta(n)$和$\Theta(n^2)$。
3. 将下列类型按有效性递减顺序排列：$\Theta(n^2)$、$\Theta(\lg n)$、$\Theta(n)$和$\Theta(n^3)$。
4. 考虑下面的问题和建议答案。看看建议的答案是正确还是错误。为什么？

   问题：假设一个盒子里有3张卡片，其中一张两面都涂成黑色，另一张两面都涂成红色，第三张一面涂成黑色，另一面涂成红色。抽出其中一张卡片，只允许看一面，那么另一面与你所看到的颜色相同的概率有多大？

   猜测答案：1/2。假设你看到的卡片的那一面是红色的（如果是黑的，讨论结果也是一样的）。只有两张卡片有红色的一面，因此你看到的卡片必是这两张中的一张。这两张中的一张背面也是红色，另一张背面就是黑色，因此你看到的卡片背面是红色的概率和是黑色的概率一样大。
5. 下面的程序段用来计算两个正整数（一个除数，一个被除数）的商（不考虑余数），方法是计算从被除数中可以减去除数，直到比除数小时减的次数。例如，7/3对应的结果应该为2，因为3可以从7中减两次。这个程序正确吗？证明你的结论。

```
Count←0;
Remainder← Dividend;
repeat ( Remainder ← Remainder - Divisor;
     Count ← Count +1 )
until ( Remainder < Divisor )
Quotient ← Count.
```

   （Count、Remainder、Dividend、Divisor和Quotient分别表示计数、余数、被除数、除数和商。）

6. 下面的程序是通过累计X个Y的总和的方法来计算非负整数X和Y的积。也就是说，3乘以4就是计算3个4的总和。下面这段程序对吗？证明你的结论。

```
Product ←Y;
Count←1;
while ( Count < X ) do
 ( Product←Product + Y;
  Count←Count +1 )
```

7. 假设前提条件是N的值是一个正整数，建立一个循环不变式，使得若下面的例程终止，Sum被赋值为0+1+2+…+N。

```
Sum←0;
K←0;
while ( K<N ) do
  (K←K+1;
  Sum←Sum+K)
```

讨论该程序终止时的真正结果。

8. 假设一个程序以及执行它的硬件都已被形式化地验证过是准确的，那么这样就能保证准确性吗？

复习题

1. 给出一组步骤的例子，使它符合5.1节首段落中给出的算法非正式定义，但不符合5.1节中给出的正式定义。
2. 解释被提议的算法的歧义性和算法表示的歧义性的区别。
3. 描述如何使用原语来帮助消除算法表示中的歧义性。
4. 选择一个你比较熟悉的学科，并设计一种伪代码来描述该学科。其中，要描述你要使用的原语以及用于表示它们的语法。（如果想不出一个科目，可以考虑体育、艺术或者工艺等。）
5. 下面的程序从严格意义上讲表示一个算法吗？为什么？

```
Count ← 0;
while ( Count not 5 ) do
  (Count←Count+2)
```

6. 从什么意义上讲，下列3个步骤并不构成一个算法？

   第1步：在直角坐标系中从点(2,5)到点(6,11)之间画一条直线。

   第2步：在直角坐标系中从点(1,3)到(3,6)之间画一条直线。

   第3步：以上面两条线的交点为中心，画一个半径为2的圆。

7. 用`repeat`结构代替`while`结构重写下面的程序段，确保它能够输出与原程序相同的值。

```
Count ← 2;
while ( Count<7 ) do
 ( 打印赋给Count的值并且
  Count←Count+1)
```

8. 利用while结构代替repeat结构重写下面的程序段，确保它能够输出与原程序相同的值。

```
Count ← 1;
repeat
 ( 打印赋给Count的值并且
  Count←Count+1 )
until ( Count=5 )
```

9. 要把以

   repeat (...) until (...)

   形式表达的后测试循环转换为以

   do (...) while (...)

   形式表达的等价的后测试循环，怎样进行？

10. 设计一个算法，对于数字0, 1, 2, 3, 4, 5, 6, 7, 8, 9的一个排列，使它能够产生一个新的排列使得其数值在这些数字所有可能的排列中仅比原排列的数值大(或者报告不存在更大的排列)，因此5647382901算法将产生5647382910排列。
11. 设计一个算法，来找出一个正整数的所有因子。例如，对于整数12，该算法得到1、2、3、4、6和12。
12. 设计一个算法，来计算从1700年1月1日起的任意一天是星期几。例如，2001年8月17日是星期五。
13. 正式程序设计语言和伪代码的区别是什么？
14. 语法和语义之间的区别是什么？
15. 下面是一个传统的十进制加法，每个字母表示不同的数字。问这些字母表示什么数字？你是怎样“入门”的？

```
  XYZ
+ YWY
-----
 ZYZW
```

16. 下面是一个传统的十进制乘法，每个字母表示不同的数字。问这些字母表示什么数字？你是怎样“入门”的？

```
   XY
 × YX
 ----
   XY
  YZ
 ----
  WVY
```

17. 下面是一个二进制加法，每个字母表示不同的二进制数字。问哪个字母表示1，哪个字母表示0？请为解决此类问题设计一个算法。

```
  YXX
+ XYX
-----
 XYYY
```

18. 有4位采矿者，他们只有一个手电筒，并且必须走过挖煤的坑道。他们最多可以两个人一起通过，并且其中一个人必须拿着手电。这4位采矿者分别叫Andrews、Blake、Johnson和Kelly，他们单独通过坑道的时间分别是1 min、2 min、4 min和8 min。当两个人一起通过坑道时，要以速度慢的人的速度为准，如何安排才能使这4人在15 min内通过坑道？解释你是怎样"入门"的？
19. 有两个酒杯，一大一小，先往小酒杯中倒满酒，再把小酒杯中的酒倒入大酒杯，然后把小酒杯中倒满水，再把小酒杯中的水倒入大酒杯。在大酒杯中均匀搅拌，再把混合液体倒回到小酒杯，直到倒满，请问此时大酒杯中的水和小酒杯中的酒哪个多？解释你是怎样"入门"的？
20. 两只蜜蜂，一只叫罗密欧，一只叫朱丽叶，它们住在不同的蜂房，但是相爱了。在一个无风的春天早晨，它们同时离开各自的蜂房来相会，它们相遇的地点在距离最近的蜂房50 m的地方，但它们都没看到对方，因此继续按各自的方向飞，直到飞到对方的蜂房，用了同等的时间发现对方没有在家，并开始返回。在距离最近蜂房20 m的地方，它们又相遇了，这次它们看到了对方，愉快地去野餐了。请问这两个蜂房的距离是多少？你是如何"入门"的？
21. 设计一个算法，给定两个字符串，检查第一个字符串是否是第二个字符串的子串？
22. 下面这个算法用来打印已知的斐波那契序列的开始部分，请标识这个循环体。哪儿是循环控制的初始化步骤？哪儿是修改步骤？哪儿是测试步骤？产生的数字列表是什么？

```
Last←0;
Current←1;
while(Current<100)do
  (打印赋给Current的值;
   Temp←Last;
   Last←Current；并且
   Current←Last+Temp)
```

23. 在下面的算法中，如果输入值分别以0和1开始，显示的数的序列是什么？

```
procedure MysteryWrite (Last, Current)
if ( Current<100 ) then
  ( 打印赋给Current的值;
  Temp←Current+Last;
  应用MysteryWrite到值Current和Temp )
```

24. 修改上一问题中的过程MysteryWrite，使得显示的数的序列次序相反。
25. 如果用二分搜索算法（图5-14）从给定的字母列表A、B、C、D、E、F、G、H、I、J、K、L、M、N、O查找出J，那么哪些字母会被查到？如果要查找Z呢？
26. 一般来说，用顺序搜索法在有6000项的列表中搜索，目标值与列表项进行比较的平均次数是多少？如果用二分搜索法呢？
27. 确定下列每个循环语句的终止条件。

```
a. while(Count<5)do(          )
b. repeat(          )
     until(Count=1)
c. while((Count<5)and(Total<56))do
     (          )
```

28. 标识下列循环结构的循环体，计算它执行了多少次。如果把测试条件改变为while(Count not 6)，会有什么情况发生？

```
Count ← 1;
while ( Count not 7 ) do
 ( 打印赋给Count的值并且
  Count←Count+3 )
```

29. 如果在计算机上执行下面这个程序，你觉得可能会发生什么问题？（提示：想想浮点算法可能导致的溢出问题。）

```
Count←one-tenth;
repeat
  ( 打印赋给Count的值并且
  Count←Count + one-tenth )
until(Count equals 1)
```

30. 设计一个递归的欧几里得算法（参见5.2节的第3题）。
31. 假设在Test1和Test2(下面已经定义了)输入值1，那么两个程序的输出结果有什么区别？

```
procedure Test1 ( Count )
if ( Count not 5 )
  then ( 打印赋给Count的值并且将Test1应用于
         Count+1 )
procedure Test2 ( Count )
if ( Count not 5)
  then ( 将Test2应用于Count +1并且打印赋给
         Count的值 )
```

32. 确定上一题中的例程的控制机制中的重要组成要素。具体而言，什么条件会使这一过程终止？过程的哪一部分可以修改终止条件？哪一部分完成过程的初始化？

33. 确定下面递归过程的终止条件。

```
procedure XXX(N)
if ( N=5 ) then ( 将XXX过程应用于N + 1)
```

34. 将下面的过程MysteryPrint的输入值设为3，记录打印的值。

```
procedure MysteryPrint (N)
if ( N>0 ) then ( 打印N的值并将MysteryPrint 过
                  程应用于N-2)
打印N + 1的值
```

35. 将下面的过程MysteryPrint的输入值设为2，记录打印的值。

```
procedure MysteryPrint (N)
if ( N>0 )
   then(打印N的值并将MysteryPrint过程应用于
        N-2)
   else(打印N的值并且
        if ( N>-1 )
           then (将MysteryPrint过程应用于
                 N+1 ))
```

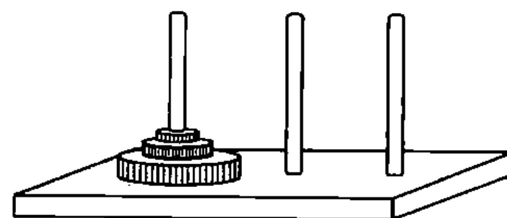
36. 设计一个算法，来（按递增顺序）生成其素数因子为2和3的正整数的序列。也就是说，你的程序应该产生这样的序列：2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27,…。在严格意义上讲，你的程序表示一个算法吗？

37. 按照列表Alice、Byron、Carol、Duane、Elaine、Floyd、Gene、Henry和Iris，回答下列问题。
    a. 哪种搜索算法(二分法或顺序法)查找Gene更快？
    b. 哪种搜索算法(二分法或顺序法)查找Alice更快？
    c. 哪种搜索算法（二分法或顺序法）能够比较快地检测出名字Bruce不存在？
    d. 哪种搜索算法（二分法或顺序法）能够比较快地检测出名字Sue不存在？
    e. 如果用顺序搜索方法查找Elaine，会进行多少次比较？如果用二分搜索呢？

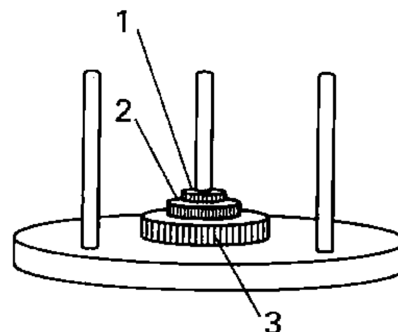
38. 0的阶乘定义为1。正整数的阶乘定义为整数本身和比自己小的非负整数的阶乘。我们用记号$n!$来表示整数n的阶乘，也就是说3的阶乘（写作3!）是$3\times(2!)=3\times(2\times(1!))=3\times(2\times(1\times(0!)))=3\times(2\times(1\times1))=6$。请设计一个递归算法来计算任意整数的阶乘。

39. a. 假设必须给一个有5个名字的列表排序，而且已经有一个算法能给含4个名字的列表排序。请利用已经设计好的算法来设计一个能给有5个名字的列表排序的算法。
    b. 基于问题a中使用的技术，设计一个能给任意长的名字列表排序的递归算法。

40. 称为汉诺塔的难题有3根柱子，每个柱子都可以放置若干个大小不同的环，这些环自底向上直径越来越小。这个问题是，如何将一个柱子上排列好的环移到另一个柱子上，规则是每次只能移动一个环，较大的环不能放在较小的环上面。我们看到，如果总共就只有一个环，那么问题就非常容易。其次，当要移若干个环的时候，如果你把除了最大的环之外的所有环都搬到另一个柱子上，那么就可以把这个最大的环搬到第三根柱子上，然后把其余的环搬到它上面。利用这个分析，开发一个递归算法来解决任意环数的汉诺塔问题。



41. 解决汉诺塔问题的另外一个方法是把3根柱子想象成一个圆圈排列，每根柱子在4点钟、8点钟、12点钟的位置上。开始时，一根柱子上的环从小到大以1, 2, 3等依次编号。最小的环编号为1。看一根柱子上面的环，如果它的编号是奇数，允许它按照顺时针方向移到下一根柱子上；如果它的编号是偶数，则允许它按照逆时针方向移到下一根柱子上（只要不把较大的环放在较小的环的上面）。在这个限制条件下，当几个柱子上有可搬的环时，总是搬编号最大的环。按照这个思路，开发一个非递归算法来解决汉诺塔问题。



42. 开发两个算法，用来打印一个工人30天期间的日薪，要求一个算法基于循环结构，另一个基

于递归结构。这个工人每天的工资是前一天的两倍（第一天的工资设为1便士）。如果在计算机上实现你的算法，那么在数的存储上面会遇到什么问题？

43. 开发一个算法来求一个正数的平方根。开始时，把这个正数本身作为根的第一个猜测值，以后按下列方法重复地产生新的猜测值：原正数除以现在的猜测值得到商，取这个商和该猜测值的平均值作为下一个猜测值。分析对这个重复过程的控制，特别是，重复的终止条件是什么？

44. 设计一个算法，列出一个由5个不同字符组成的字符串中的字符的其他可能的排列。

45. 设计一个算法，在给定的名字列表中找到最长的名字。如果列表里有多个“最长”的名字，那么算法应如何解决？特别是，如果列表里的所有名字的长度都一样，算法又应如何解决？

46. 设计一个算法，对于一个有5个或更多表项（数）的列表，在不对整个列表进行排序的情况下，找出5个最小的和5个最大的表项。

47. 对名字Brenda、Doris、Raymond、Steve、Timothy和William进行排序，要求在使用插入排序算法（图5-11）进行排序时比较次数最少。

48. 对于有4000个名字的列表，使用二分搜索算法（图5-14）时最多检查多少个表项？使用顺序搜索算法（图5-6）呢？试对二者进行比较。

49. 使用大Θ记号对传统的小学加法和乘法的算法进行分类。也就是说，如果两个有n个数字的数相加，那么要做多少次一位的加法？如果两个有n个数字的数相乘，那么要做多少次一位的乘法？

50. 有时对一个问题稍作变动就可能使它的解的形式发生重大改变。例如，设计一个简单的算法来解决下述问题，并用大Θ记号进行归类：

    把一群人分为两个小组（人数不限），使得两个小组成员的年龄的总和的差尽可能大。

    现在把问题改为，使得两个小组成员的年龄的总和的差尽可能小，再利用大Θ记号进行归类。

51. 从下面的列表中找出一组数，使其总和等于3165。你的解法效率如何？

    26, 39, 104, 195, 403, 504, 793, 995, 1156, 1677

52. 下面例程中的循环会终止吗?解释你的回答。如果这个例程实际在一台计算机上执行（见1.7节），说明可能会发生的情况。

```
X←1;
Y←1/2;
while ( X不等于0 ) do
  ( X←X - Y;
   Y←Y÷2 )
```

53. 下面的程序段用来计算两个非负整数X和Y的乘积，方法是累计X个Y的和。也就是说，3乘以4是通过累计3个4得到。这个程序段正确吗？为什么？

```
Product←0;
Count ←0;
repeat(Product← Product + Y,
     Count← Count +1)
until(Count = X)
```

54. 下面的程序段是用来报告正整数X和Y中哪个大的，这段程序正确吗？为什么？

```
Difference ← X-Y;
if ( Difference是正数)
  then (print "X is bigger than Y")
  else (print "Y is bigger than X")
```

55. 下列的程序段用来从一个非空的整数列表中找到最大的项。这个程序段正确吗？为什么？

```
TestValue ← first list entry;
CurrentEntry ←first list entry;
while ( CurrentEntry 不是最后一项 ) do
  (if ( CurrentEntry > TestValue )
      then ( TestValue←CurrentEntry )
   CurrentEntry←下一个表项)
```

56. a. 标识图5-6表示的顺序搜索算法的前提条件。为这个程序里的while结构确定一个循环不变式，当它与终止条件结合时，就意味着，在该循环终止时该算法将正确地报告成功或失败。

    b. 给出一个论据说明图5-6里的while循环事实上是会终止的。

57. 基于赋给X和Y的值是非负整数的前提条件，标识下述的while结构里的循环不变式，当它与终止条件结合时，就意味着，与Z相联系的值在循环终止时一定是X-Y。

```
Z←X;
J←0;
while ( J<Y ) do
  ( Z←Z-1;
   J←J+1)
```

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 现在完全验证复杂程序的正确性几乎是不可能的。在何种情况下（如果有这种情况），程序的开发者对错误负有责任?
2. 假设你有一个很好的想法，并把它开发成了一个为很多人所用的产品，而这已经耗费了你一年的时间和50 000美元。可是，该产品的最终形式可能被许多没有向你购买该产品的人所使用。为了获得补偿你具有哪些权利？盗版计算机软件合法吗？音乐和电影呢？
3. 假设一个软件包非常昂贵，超过了你的预算，那么复制这个软件供自己使用是否有违道德?（毕竟，因为你无论如何都不可能去购买这个软件包，所以对供应商的销售额不会有影响。）
4. 人们对河流、森林、海洋等的所有权一直争论不休，那么在什么意义上应该给某人或某机构一个算法的所有权？
5. 有些人觉得新算法是被发现的，而另一些觉得新算法是被创建的。你同意哪种说法？这些不同观点会导致关于算法的所有权和一般所有权的不同结论吗？
6. 设计一个实现非法行为的算法是道德的吗?它与该算法是否被实际执行有关吗？开发出这种算法的人具备该算法的所有权吗？如果具备，那个人应该拥有哪些权利？算法的所有权应该与该算法的目的有关吗？大肆宣扬和散布破解安全的技术是道德的吗？它与破解的内容有关吗？
7. 一个作家会获得为一部小说支付的电影版权费，尽管这个故事在电影版本中经常被改动。一个故事要改变成一个不同的故事，它必须改变多少呢？对于算法来说，一个算法要变成一个不同的算法，必须要对这个算法做多少改动呢？
8. 面向18个月或更小儿童的教育软件现在正在销售。支持者认为，这些软件提供的一些图像和声音是许多孩子无法通过其他途径获得的，反对者认为，它是父母子女之间交流的拙劣的替代品。你有什么看法？你应该在没有对这软件了解更多的情况下采取行动吗？如果是，你会怎么做？

课外阅读

Aho, A. V. ,J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Boston, MA: Addison-Wesley, 1974.

Baase, S. *Computer Algorithms: Introduction to Design and Analysis,* 3rd ed. Boston, MA: Addison- Wesley, 2000.

Barnes, J. *High Integrity Software: The SPARK Approach to Safety and Security*. Boston, MA: Addsion-Wesley, 2003.

Gries, D. *The Science of Programming*. New York: Springer-Verlag, 1998.

Harbin, R. *Origami-the Art of Paper Folding*. London: Hodder Paperbacks, 1973.

Johnsonbaugh, R. and M. Schaefer. *Algorithms*. Upper Saddle River, NJ: Prentice-Hall, 2004.

Kleinberg , J. and E. Tardos. *Algorithm Design*. Boston, MA: Addison-Wesley, 2006.

Knuth, D. E. *The Art of Computer Programming*, vol.3, 3rd ed. Boston, MA: Addison-Wesley, 1998.

Levitin, A. V. *Introduction to the Design and Analysis of Algorithms*, 2nd ed. Boston, MA: Addison-Wesley, 2007.

Polya, G. *How to Solve It*. Princeton, NJ: Princeton University Press, 1973.

Roberts, E. S. *Thinking Recursively*. New York: Wiley, 1986.

第6章 程序设计语言

本章我们来学习程序设计语言。我们的目标并不是学习一门特定的程序设计语言，而是学习与程序设计语言相关的一些知识。我们将要考查程序设计语言及其相关联的方法之间的共性和个性。

本章内容

如果人们不得不使用机器语言直接编写程序，那么要想开发像操作系统、网络软件和大型应用软件这样的复杂软件系统基本上是不可能的。我们至少可以这么说，在试图组织和设计一个复杂系统的同时，处理这些与机器语言有关的烦琐而复杂的细节必定是一项繁重的工作。因此，类似伪代码的程序设计语言开发出来了，它使得算法既方便人理解，又能够很方便地转换为机器指令。本章，我们的目标是考查计算机科学领域内这些程序设计语言的设计和实现。

6.1 历史回顾

我们从追溯程序设计语言发展的历史开始。

6.1.1 早期程序设计语言

正如在第 2 章学过的，现代计算机的程序由采用数字编码的指令序列组成。这样的编码系统称为机器语言。但是，用机器语言编写程序是一项冗长乏味的任务，而且经常出错，在工作完成之前，这些错误必须被找到和更正——这个过程称为**调试**（debugging）。

20 世纪 40 年代，研究人员为了简化程序设计过程开发了符号系统，使得指令可以用助记符表示，不再使用数字形式。例如，指令：

把寄存器 5 的内容送入寄存器 6

可用第2章介绍的机器语言表示为：

```
4056
```

而使用助记符系统时，可以表示为：

```
MOV  R5 ,R6
```

再举一个更大一点的例子，机器语言例程：

```
156C
166D
5056
306E
C000
```

这一例程将存储单元6C和6D的内容相加，将结果存入地址6E（见2.2节图2-7），可使用助记符表达为：

```
LD R5, Price
LD R6, ShippingCharge
ADDI R0, R5 R6
ST R0, TotalCost
HLT
```

[这里，我们使用了LD、ADDI、ST和HLT分别表示装入、相加、存储和停机。此外，我们用描述性名称Price、ShippingCharge和TotalCost相应表示地址为6C、6D和6E的存储单元，这些描述性的名称常称为**标识符**（identifier）]。注意，助记符形式虽然有不足之处，但是与数字形式相比，它确实能更好地表达例程含义。

建立起来这种助记符系统后，人们就开发了称为**汇编器**（assembler）的程序来将用助记符形式表达的程序转换为机器语言。以此方式，人们可以使用这种助记符开发程序，然后再用汇编器把它转换为机器语言，而不必直接使用机器语言开发程序。

表示程序的助记符系统统称为**汇编语言**（assembly language）。当汇编语言最初被开发出来时，它们代表了在研究更好的程序设计技术方面迈出了巨大一步。实际上，汇编语言的出现是革命性的事件，以至于它们被称为第二代程序设计语言，而第一代程序语言是机器语言。

尽管汇编语言与机器语言相比有不少的优势，但是仍有一些不足——它们没有提供最终的程序设计环境。毕竟，在汇编语言中使用的原语本质上和与之相对应的机器语言中的相同，这两者的不同仅仅体现在用于表示它们的语法上。因此，用汇编语言写的程序必然依赖于机器，也就是说，程序中使用的指令都是遵循特定的机器特性来编写的。用汇编语言写的程序不能方便地移植到另一种机器上，这是因为这个程序必须重写以遵循这种新机器的寄存器配置和指令系统。

汇编语言的另一个缺点是，尽管程序员不再必须使用数字形式编写代码，但仍不得不从机器语言的角度一小步一小步地思考。这种情况很类似于房屋设计——我们毕竟还是要根据木板、钉子和砖块等来设计。确实，在实际的房屋建造中，最后的确还需要一个基于这些基本元素的描述，但是如果我们根据诸如房间、窗户和门等更大一些的单元来思考设计，设计过程应该会更简单一些。

简而言之，最终构建产品所使用的基本原语不一定是在设计过程中使用的原语。这个设计过程应该更适合使用更高级的原语——每一个原语都代表了一个与产品的主要特性相关的概念。一旦设计过程结束，这些原语就能够被翻译成与实现细节相关的较低级概念。

根据这种哲理，计算机科学家开始开发比低级的汇编语言更易于开发软件的程序设计语言。结果就出现了第三代程序设计语言，它们不同于早期的程序设计语言，因为它们的原语不仅是更高级别的（它们代表比较多的指令）而且是**机器无关**（machine independent）的（它们不依赖于特定计算机的特性）。一个著名的早期程序设计语言就是FORTRAN（FORmula TRANslator），它是为科学和工程应用开发的，还有COBOL（COmmon Business-Oriented Language），由美国

海军开发，用于商业应用。

一般来说，第三代程序设计语言的方法就是标识更高级的原语的一个集合（基本上和我们在第 5 章中开发伪代码的思路一致），而软件要使用这些原语来开发。每一个原语要能够当做相对应的机器语言中的一个较低级的原语序列而被实现。例如，语句

```
assign TotalCost the value Price + ShippingCharge
```

描述了一个高级的动作，它并不与执行此任务的特定机器相关，但它可以由先前讨论过的机器指令序列来实现。因此，我们的伪代码结构

```
标识符←表达式
```

是潜在的高级的原语。

一旦这样的高级原语集合被标识出来，就可以编写出一个称作**翻译器**（translator）的程序，这个程序能够把用高级原语表示的程序翻译成机器语言程序。除了常常将一些机器指令编译为短序列来模拟一个高级原语所请求实现的动作，翻译器很类似于第二代语言的汇编程序。因此，这种翻译程序通常也称为**编译器**（compiler）。

翻译器的一种替代方案是**解释器**（interpreter），它是作为实现第三代程序设计语言的另一种方法出现的。这类程序类似于翻译器，不同之处是，它们在翻译出指令的同时执行指令，而不是把翻译出的指令记录下来供将来使用。也就是说，解释器不产生供以后执行使用的机器语言程序，而实际上是依据程序的高级形式执行它。

另一个枝节问题是，我们应当注意到发展第三代程序设计语言的任务并没有想象得那么简单。使用类似于自然语言的形式来编写程序的思想是革命性的，以至于首先在许多管理部门人员中引起了争论。第一个编译器的开发者 Grace Hopper 常常叙述这样的故事，她在演示第三代语言的翻译器时，该语言使用的是德文词汇，而不是英文词汇。问题是，程序设计语言是围绕一小组原语来构造的，而这些原语可以用各种各样的自然语言来表达，只要稍微修改翻译器。但是，她惊讶地发现许多听众对于她在第二次世界大战的许多年中一直在教计算机“理解”德语感到惊讶。今天，我们知道理解一门自然语言涉及的问题远远超过对不多几条严格定义的原语的响应。的确，**自然语言**（natural language，例如英语、德语和拉丁语）不同于**形式语言**（formal language，例如程序设计语言），后者是由语法严格定义的（见 6.4 节），而前者还远远没有涉及形式语法分析。

跨平台软件

典型的应用程序必须依赖操作系统来完成它的任务。它也许需要窗口管理程序提供的服务与计算机用户进行交互，或者需要利用文件管理程序从海量存储器中检索数据。但是，不同的操作系统可能要求以不同的方式请求这些服务。这样一来，对于需要跨网络和互联网传输和执行的程序而言，网络和互联网涉及各种设计不同的机器和不同的操作系统，因此该程序必须要做到与操作系统无关，同时与机器无关。“跨平台”这个术语用于反映这种额外的独立程度。也就是说，跨平台软件是一个可以独立于操作系统设计和具体机器硬件设计的软件，因此在整个网络上是可执行的。

6.1.2　独立并超越机器

随着第三代语言的开发，“与机器无关”的目标在很大程度上实现了。既然第三代语言中的

语句不再与某种特定的机器特性有关，它们就能够在不同的机器上被轻松编译。通过使用合适的编译器，一个用第三代语言写的程序理论上应该能够在任何机器上使用。

但是，现实并不是这样简单。当一个编译器设计出来时，目标计算机的具体特征有时候会作为要翻译的语言的条件反映出来。例如，不同的计算机处理I/O操作有不同方法，导致了“相同”的语言在不同的机器上有着不同的特性或方言。因此，对于一个程序而言，从一台机器移植到另一台机器上至少要有少量的修改，这通常是必要的。

伴随可移植性问题而来的是，对在某些情况下关于特定语言的正确定义应该包括哪些东西缺乏一致性的认识。为此，美国国家标准化学会（ANSI）和国际标准化组织（ISO）对一些使用比较普遍的语言进行整理并公布了一系列标准。对于其他情况，制定非正式标准是由于某种语言的某个版本的流行，以及其他编译器的作者实现一个兼容的产品的意愿。但是，即使是高度标准化了的语言，编译器的设计者通常还是会提供一些不包括在标准版本之中的特性，这有时也被称为语言扩展。如果一个程序员利用这些特性，他所设计的程序将不再与采用其他厂商的编译器的环境兼容。

在程序设计语言的整个历史中，由于以下两个原因，第三代语言没有真正达到与机器无关这个事实并不重要。第一，它们已经几乎达到了机器无关性，软件可以从一台机器相对比较容易地移植到另一台机器。第二，“与机器无关”的最终目的仅仅是其他更高要求的一个基础。确实，计算机能够响应像

把`Price + ShippingCharge`的值赋给`TotalCost`

这样的高级语句，这种现实致使计算机科学家们梦想实现这样的程序设计环境，它允许人们用抽象的概念与机器进行交互，而不再强迫机器把这些概念翻译成与机器兼容的格式。此外，计算机科学家更希望机器能够实现许多算法发现过程，而不是仅仅能够执行算法。结果带来程序设计语言谱系的不断扩大，以至于按照不同世代的清晰划分受到挑战。

6.1.3 程序设计范型

将程序设计语言划分为不同代，是基于一个线性尺度的（见图6-1），对于一个语言的定位则是由这门语言的使用者不受机器世界语言约束的程度，以及允许从问题的角度来考虑的程度决定的。实际上，程序设计语言的发展并不确切地遵循这种方式，而是沿着不同的可以选择的程序设计过程（称为**程序设计范型**，programming paradigm）发展。于是，图6-2所示的多路径图能更好地描述程序设计语言的发展历程，该图显示了来源于不同范型的不同路径的出现和发展。具体地说，这幅图展示了4条路径，分别代表了函数式范型、面向对象型范型、命令型范型和说明性范型，图中通过与其他语言的相对位置关系，指出了与每一个范型联系的各种语言的诞生时间。（但是这并不暗示一种语言必然是从一种早期语言中发展而来的。）

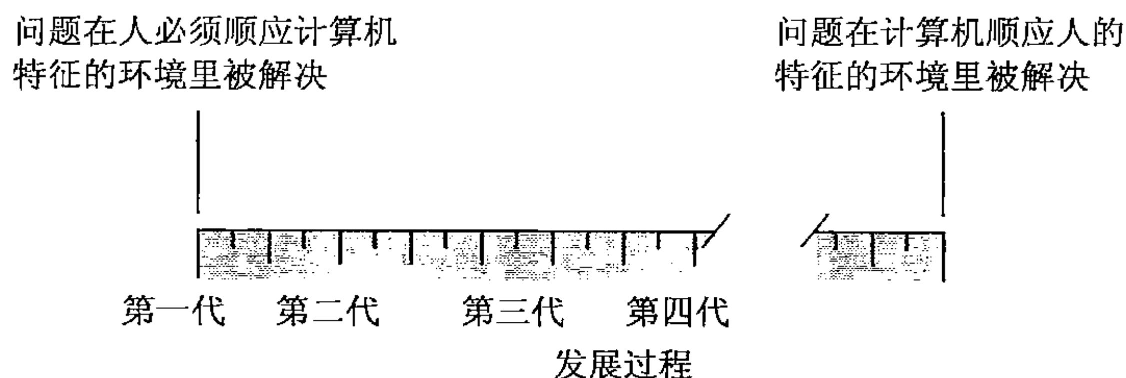


图6-1 程序设计语言的发展

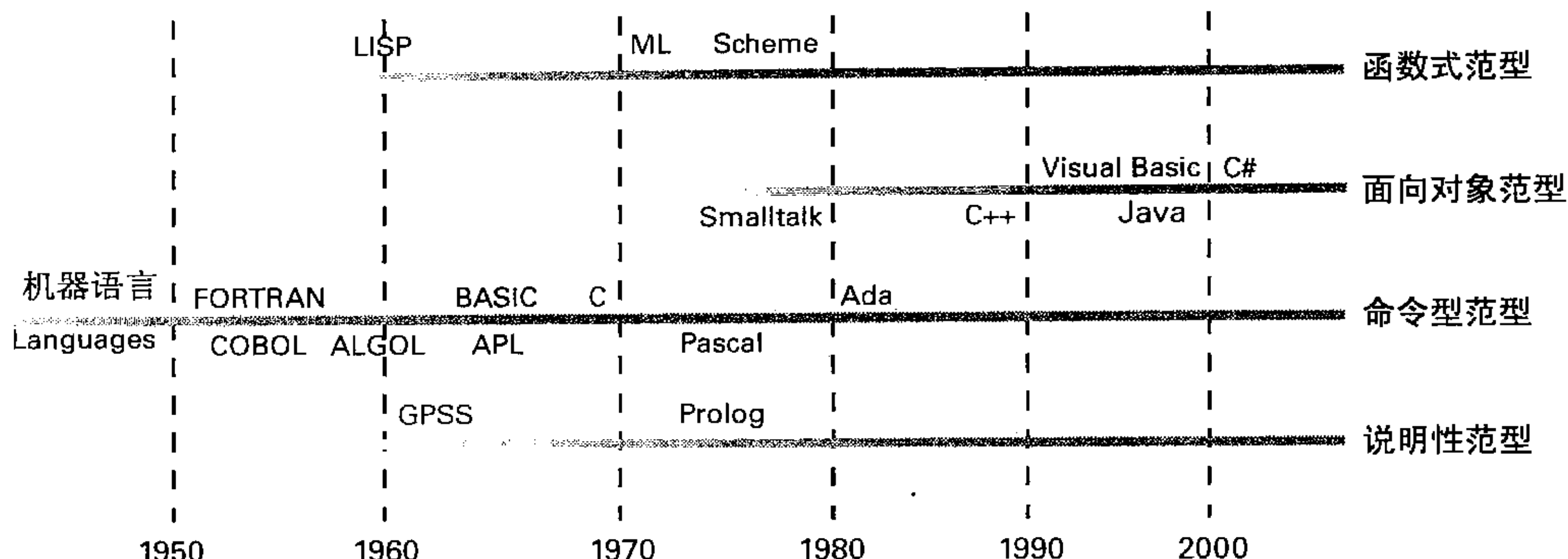


图6-2 程序设计范型的演变

我们应当注意到，尽管在图 6-2 中标识的范型称为程序设计范型，然而对不同的分支（即路线）的选择已经超出了程序设计过程的范畴。它们基本上代表了构建问题解决方案的不同方法，并且因此影响整个软件开发过程。在这种意义上，程序设计范型这个词有些使用不当，一个更现实的术语应该是软件开发范型。

命令型范型（imperative paradigm），也叫**过程范型**（procedural paradigm），它代表了程序设计过程的传统方法。命令型范型就是第 5 章中的伪代码以及第 2 章讨论的机器语言所基于的范型。正如它的名字所暗示的那样，命令型范型定义程序设计过程是开发一个命令序列，遵照这个序列，对数据进行操作以产生所期望的结果。因此命令型范型告诉我们要通过寻找解决问题的算法来处理程序设计过程，并且要将这个算法表示为命令的序列。

与命令型范型相对的是**说明性范型**（declarative paradigm），它要求程序员描述要解决的问题，而不是解决该问题的算法。更准确地说，一个说明性程序设计系统应用一个预先设定的通用的解决问题的算法来解决面临的问题。在这种环境中，程序员的工作变成了开发问题的准确陈述，而不是描述一个解决问题的算法。

在开发基于说明性范型的程序设计系统时，一个主要的障碍就是需要一个潜在的解决问题的算法。正因为这样，早期的说明性程序设计语言试图用于某些特定的用途，并满足某些特殊的应用。例如，许多年以来，说明性方法已经用于模拟一个系统（经济的、物理的、政治的等）来判定假设或获得预测。在这样的环境中，潜在的算法本质上是通过重复计算参数的值（国内生产总值、贸易赤字等）来模拟时间推移的过程，其中所用的参数都基于以前计算得到的值。于是，用于这类模拟的说明性语言需要首先实现一个执行该重复过程的算法。然后，使用这个系统的程序员唯一的任务就是描述要模拟的情况。按照这种方法，天气预报员不必开发一个预报天气的算法，只需要描述当天的天气情况，让潜在的模拟算法来产生未来几天的天气预报。

人们发现，数学里的形式逻辑学科提供了一种简单的、适用于通用的说明性程序设计系统的问题求解算法，这极大地促进了说明性范型的发展。其结果是人们对于说明性范型更加关注且**逻辑程序设计**（logic programming）出现了，这将在 6.7 节中进一步讨论。

另一种程序设计范型是**函数式范型**（functional paradigm），基于该范型的程序可以被看做是接受输入和产生输出的实体。数学家将这样的实体称为函数，这就是这种范型被称为函数式范型的原因。函数式范型的程序由连接预先定义的小的程序单元（预定义的函数）构建而成，其中每一个程序单元的输出可以用来作为另一个程序单元的输入，通过这种方式可以获得所期望的整体上的输入-输出关系。简而言之，这种函数式范型的程序设计过程就是把函数构造成简单函数的嵌套联合体。

举一个例子，图 6-3 说明了如何由两个较简单的函数构成计算支票簿余额的函数。其中一个称为 Find_sum，它接收一些值作为输入，产生这些值的和作为输出。另一个称为 Find_diff，它接收两个值，计算它们的差。使用 LISP 程序设计语言（一个著名的函数式程序设计语言）时，图 6-3 所示的结构可以用下列表达式表示：

```
(Find_diff (Find_sum Old_balance Credits) (Find_sum Debits))
```

表达式的这个嵌套结构（如括号中指定的）反映了这样一个事实，即函数 Find_diff 的输入是由 Find_sum 的两次应用产生的。Find_sum 的第一次应用的结果是所有的 Credits 加到 Old_balance 上，Find_sum 的第二次应用就是计算所有的 Debits 的总和。然后，函数 Find_diff 使用这两个结果以得到新的支票余额。

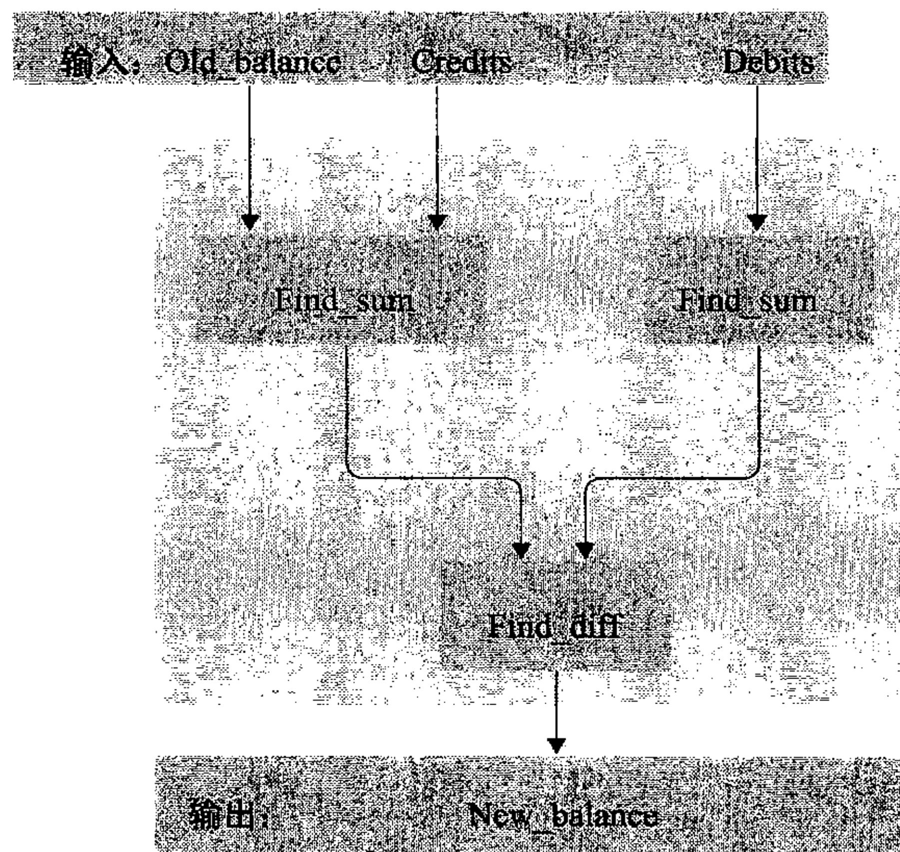


图6-3 由较简单的函数构造支票簿余额计算函数

为了更全面理解函数式范型与命令型范型之间的区别，我们把求支票簿余额的函数式程序同下面遵循命令型范型的伪代码程序进行一下比较：

```
Total_credits←sum of all Credits
Temp_balance←Old_balance + Total_credits
Total_debits←sum of all Debits
Balance←Temp_balance - Total_debits
```

注意，这个命令型程序由多条语句组成，每条语句都要求执行计算，并请求把这个结果存储起来供以后使用。与命令型程序不同，函数式程序由单个语句组成，程序中的每个计算结果都会立即传送到下一个函数式程序。从某种意义上说，命令型程序可以看做是若干工厂的集合，每个工厂把原材料生产成产品，并把这些产品存放在仓库里。然后，产品从这些仓库被装运到其他需要这些产品的工厂。函数式程序与它不同，类似于许多工厂的集合，在这个工厂的集合里，各个工厂协调一致，每个工厂仅仅生产其他工厂订购的产品，然后立刻把这些产品运送到目的地而不需要中间仓库。这种效率也是函数式范型的支持者声明的优点之一。

还有另一种程序设计范型（当今的软件开发领域中最著名的一个）是**面向对象范型**（Object-Oriented Paradigm），它是与称为 OOP（Object-Oriented Programming，面向对象程序设计）的程序设计过程相联系的。遵照该范型，一个软件系统被看做是**对象**（object）的集合，

每一个对象都能够执行与自己直接相关的动作以及其他对象请求的动作。总之，这些对象通过交互解决问题。

再举一个面向对象方法的例子，考虑一个开发图形用户界面的工作。在面向对象环境中，屏幕上的图标将作为对象来实现。每个对象包含了一组过程（在面向对象环境中称为**方法**，method），这些过程描述了对象是如何响应各种事件的发生的，诸如被鼠标单击选中或者是被鼠标在屏幕上拖动等。因此，整个系统是对象的集合，每一个对象都知道如何响应与之有关的事件。

为了比较命令型范型与面向对象范型，这里考虑一个涉及名字列表的程序。在传统的命令型范型中，这个列表仅仅被看为数据的一个集合，任何一个访问这个列表的程序必须包括执行所需操作的算法，而在面向对象方法中，这个列表将被构建成由列表和操作这个列表的方法的集合组成的对象。（对这个列表的操作可能包括插入、删除表项、判断表是否为空，以及为列表排序等。）因此，另外一个需要操作这个列表的程序单元不再包含执行这些任务的算法，而是要利用这个对象中提供的过程。从某种意义上说，程序要求列表自己把自己排好序，而不是像在命令型范型中那样对列表排序。

尽管我们将要在6.5节更详细地讨论面向对象范型，但面向对象范型在当今软件开发领域的重要性要求我们在这里引入类的概念。回忆可知，一个对象可以包含数据（如名字列表），同时包含完成操作的方法的集合（如在列表中插入新的名字）。这些特征必须通过所写的程序中的语句来描述。对象的属性的这一描述称为**类**（class）。一旦类被构造好了，它就可以在任何需要具有这些特性的对象的时候被使用。因此，几个对象可以基于同一个类（即由同一个类构建）。像同卵双胞胎一样，由于这些对象产生于相同的模板（相同的类），它们虽然具有相同特征但却是不同的实体。因此，一旦一个类构建好以后，在任何需要包含该类的特征的对象的时候都可以重用。基于特定的类构建的对象称为这个类的**实例**（instance）。

由于对象是明确定义的单元，在可重用的类中，其描述是孤立的，所以面向对象范型受到了欢迎。进而，面向对象程序设计的支持者指出面向对象范型为软件开发的“构建块”方法提供了一个很自然的环境。他们设想了预定义的类的软件库，通过这个库，新的软件系统能够像许多传统的产品构建于现成的组件一样构建出来。构建和扩展这样的库是一个持续的过程，我们将在第7章学习到。

最后，我们应该注意到，包含在一个对象内的方法实质上是一些小的命令型程序单元。这就意味着，大多数基于面向对象范型的程序设计语言都包含许多可以在命令型语言中找到的特性。例如，当前比较流行的面向对象语言 C++就是通过在 C 语言这个命令型语言中添加一些面向对象的特性开发出来的。此外，从 C++派生出来的 Java 和 C#也都继承了命令型语言的精髓。在 6.2 节和 6.3 节，我们将探究命令型语言的许多特性，在这样做的同时，我们将讨论贯穿在今天绝大多数面向对象软件里的概念。然后，在 6.5 节中，我们将学习面向对象范型专有的特性。

问题与练习

1. 从什么意义上说用第三代语言编写的程序是与机器无关的？从什么意义上说它们还是依赖于机器的？
2. 汇编器和编译器的区别是什么？
3. 我们可以用下面的话概述命令型程序设计范型：它强调的是描述一个可以导致解决问题的过程。请给出说明性范型、函数式范型和面向对象范型的类似概述。
4. 在什么意义上，第三代程序设计语言比早期的程序设计语言更高级？

6.2 传统的程序设计概念

在本节中，我们将研究命令型程序设计语言和面向对象程序设计语言中的一些概念。我们将会从Ada、C、C++、C#、FORTRAN和Java等程序设计语言中引出一些例子。我们的目标并不是针对某一种语言的细节纠缠不放，而只是要展示常见语言特性如何出现在实际的程序设计语言中。因此，我们的程序设计语言集合只是具有代表性的例子。C是第三代命令型语言，C++是通过对C语言进行扩展得到的面向对象的程序设计语言，Java和C#均继承了C++的一些特性，它们都是面向对象语言。（Java是SUN公司开发的，而C#是由微软公司开发的；SUN公司后来被Oracle公司收购）。FORTRAN和Ada最初是作为第三代命令型语言设计的，尽管它的最新版本包含了大多数的面向对象范型。附录D简短地介绍了这些语言中每一种的背景。

尽管我们在例子中涉及了诸如C++、Java、C#之类的面向对象语言，但是本节中将把程序想象成是基于命令型范型编写的程序，这是因为面向对象程序（描述一个对象是怎样响应外部激励的过程等）中的许多单元基本上都是简短的命令型语言程序。在6.5节，我们将主要讨论面向对象范型的独有特性。

脚本语言

一部分命令编程语言是称为**脚本语言**（scripting language）的语言集合。这些语言通常用来执行管理任务，而不是开发复杂的程序。这种任务的表述称为**脚本**（script），它解释了术语"脚本语言"。例如，计算机系统的管理员也许会写一个脚本来描述一系列每晚执行的需要保持记录的活动，而PC的用户也许会写一个脚本来指导一系列程序的执行，以从数码相机中读取照片，通过日期对照片建立索引，以及在档案存储系统中存储照片的副本。脚本语言的起源可以追溯到20世纪60年代的作业控制语言，当时在批处理作业的调度中它被用于指导操作系统（参看3.1节）。甚至在今天，许多人都认为脚本语言是指导其他语言执行的语言，这就对现在的脚本语言的认识产生了局限性。脚本语言的例子包括Perl和PHP，二者在控制服务器Web应用（参看4.3节）和VBScript中都很常见，VBScript是Visual Basic的非标准语言，它是由微软开发的并用于Windows的特定环境下。

通常，程序由一组语句组成，这些语句一般可以分成3类：声明语句、命令语句和注释。**声明语句**（declarative statement）定义了在程序中使用的需要自定义的术语，如用来引用数据项的名称；**命令语句**（imperative statement）描述了潜在的算法里的步骤；**注释**（comment）则通过比较人性化的形式来解释程序中的一些复杂特性，从而提高了程序的可读性。通常，命令语言的程序（或者面向对象程序中的命令型语言程序单元）可以被认为具有图6-4描述的结构。它以描述程序所操作的数据的一组声明语句开始，紧接其后的是描述被执行的算法的命令语句（图6-4）。现在，很多语言都允许声明语句和命令语句自由交织存在，但其概念上的区别依然存在。注释语句是很分散的，仅仅出现在需要对程序进行解释的地方。

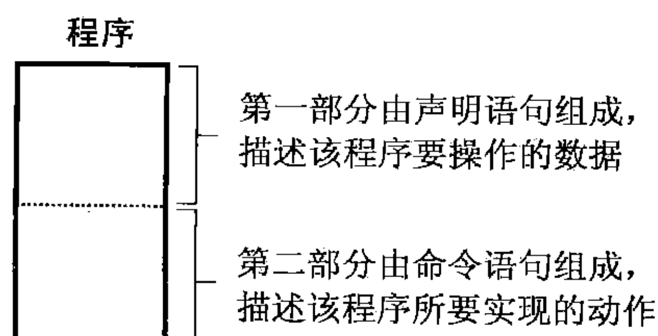


图6-4 一个典型的命令型程序或程序单元的结构

根据指引，我们通过语句目录来进行编程概念的研究，该语句目录的顺序是我们在一个程序中可能遇到的这些语句的顺序，以与声明语句有关的概念开始。

6.2.1　变量和数据类型

正如在6.1节中提到的那样，高级程序设计语言允许使用描述性的名字指代存储器地址，而不必再使用数字地址，这样的名字称为**变量**（variable）。之所以这样取名是因为，随着程序的执行，只要改变了存放在这个存储单元里的值，那么与该名字相联系的值就改变了。在程序使用这个变量之前，我们的示例语言要求必须通过一个声明语句建立变量。同时，声明语句也会要求程序员描述变量所指代的将存储在存储器地址中的数据的类型。

这样的类型称为**数据类型**（data type），它决定了数据的编码方式以及在该数据上可执行的操作。例如，**整型**就是可能以二进制补码形式存储的数值型数据，它是由全体整数组成的。可以在整型数据上进行的操作包括传统的算术运算和比较运算，如判断一个数是否比另一个数大。**实型**（real）有时也称为**浮点型**（float），是指可能以浮点形式存储的整数之外的数值型数据。可以在实型数上进行的操作很类似于那些可以在整型数上进行的操作，但是注意，把两个实型数相加与把两个整型数相加是两个不一样的操作。

假设我们需要在一个程序中使用变量 `WeightLimit` 来指代主存中以二进制补码形式编码的数据值的地址。在程序设计语言C、C++、Java和C#中，我们可以在程序的头部插入声明语句：

```
int  WeightLimit;
```

这个语句的意思是："名字`WeightLimit`将要在后面的程序中用到，它指代一个以二进制补码记数法表示的存放在存储器某个区域的值。"同一类型的多个变量通常在同一个声明语句中声明。例如，语句

```
int  Height, Width;
```

声明了两个整型变量`Height`和`Width`。此外，大多数语言允许在变量声明时，为变量赋一个初始值。因此，语句

```
int  WeightLimit = 100;
```

不仅声明了一个整型变量`WeightLimit`，而且还为这个变量赋了一个初始值`100`。

其他通用数据类型还包括字符型和布尔型。**字符型**（character）指的是由符号组成的数据，它们通常使用ASCII码或者Unicode字符集进行编码存储。可以在这种数据上进行的操作包括比较运算，如按照字母顺序判断一个字符是否在另一个字符的前面；判断一个字符串是否是另一个字符串的子串，以及将一个字符串连接在另一个字符串的尾部从而形成一个更长的字符串等。语句

```
char Letter, Digit;
```

在程序设计语言C、C++、C#和Java中用来声明两个字符型变量：`Letter`和`Digit`。

布尔型（Boolean）是仅仅有真和假两个值的数据类型。可以在布尔型数据上进行的操作包括判断当前的值是真还是假。例如，如果变量 `LimitExceeded` 被声明为一个布尔型变量，那么下面这种形式的语句：

```
if  (LimitExceeded)  then  (...)  else (...)
```

是很合理的。

作为原语包括在程序设计语言里的数据类型（像对于整型的`int`，对于字符的`char`）称为**基本数据类型**（primitive data type）。我们所知的整型、实型/浮点型、字符型、布尔型是通用的原语，其他数据类型（包括图像、音频、视频以及超文本）目前还没有成为程序设计语言的通

用原语。但是，像 GIF、JPEG 和 HTML 这样的类型可能马上就要像整型和实型一样通用。在 6.5 节和 8.4 节，我们将学习到面向对象范型如何使程序员在一门编程语言提供的原始数据类型基础之上扩展可用的数据类型。的确，这种能力是面向对象范型被称赞的特性。

下面程序段是用 C 语言及其派生语言 C++、C#和 Java 表达的声明语句。变量 `Length` 和 `Width` 声明为实型 / 浮点型，变量 `Price`、`Tax` 和 `Total` 声明为整型，变量 `Symbol` 声明为字符型。

```
float  Length, Width;
int    Price, Tax, Total;
char   Symbol;
```

在 6.4 节，我们会看到翻译器如何利用从这些说明语句中收集到的知识，把一个程序从高级语言形式翻译为机器语言形式。这里，我们要注意，这些信息可以用来识别错误。例如，对于两个早先声明为布尔类型的变量，如果翻译器发现一个要求对它们做加法的语句，那它很可能认为这个语句是错误的，并把这个结果报告给用户。

6.2.2 数据结构

除了数据类型，程序中的变量通常与**数据结构**（data structure）相联系，即数据在概念上的形态与布局相联系。例如，文本通常被看做是一个长的字符串，而销售记录可能被看为数字值的矩形表，其每一行代表了某位销售人员完成的销售，而每一列代表了某一天所完成的销售。

一个常用的数据结构是**数组**（array），即一块相同类型元素组成的数据块，如一维表、一个由行和列组成的二维表或更高维数的表。为了在程序中建立这样的数组，大多数程序设计语言要求声明语句在声明数组名字的同时也要明确指出数组每一维的长度。例如，图 6-5 显示了由 C 语言语句

```
int  Scores[2][9];
```

声明的概念上的结构，它的意思是变量`Scores`将要在后面的程序中使用到，并且是一个有2行和9列的二维的整型数组，而在FORTRAN中同样的声明语句要写成

```
INTEGER Scores(2,9)
```

一旦声明了一个数组，就能够通过它的名字在程序中的任何地方引用它，或者通过一个称作**索引**（index）的整数值来标识这些数组的组成元素，索引明确了行、列等所需的信息。但是，索引的范围在不同的语言中是不同的。例如，在C、C++、Java和C#语言中，索引从0开始，也就是说对`Scores`数组（上文已定义）的第2行第4列的项应该用`Scores[1][3]`来引用，而第1行第1列的项应该用`Scores[0][0]`来引用。相反，在FORTRAN程序中索引是从1开始的，所以第2行第4列的项对应于`Scores[2][4]`（可再参考图6-5）。

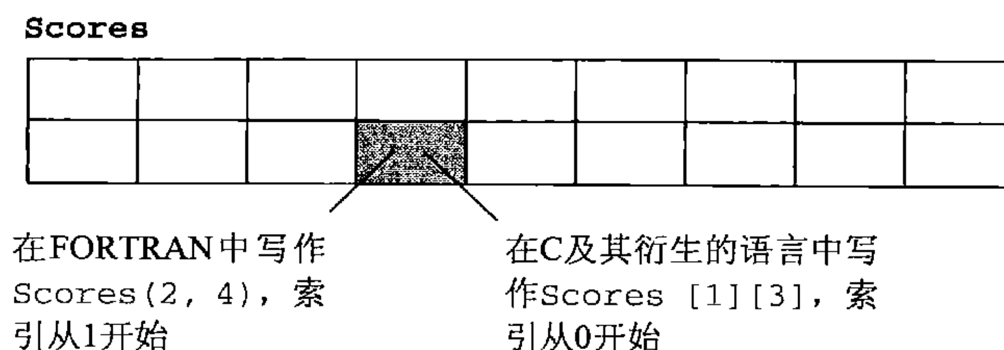


图6-5 拥有2行9列的二维数组

相比由同一种数据类型的数据元素组成的数组，**聚合类型**［aggregate type，也称结构（structure）、**记录**（record），有时还称**异构数组**（heterogeneous array）］是其元素可能具有不同类型的数据块。例如，一个雇员的数据块也许包括一个字符型的 Name、一个整型的 Age 以及一个实型的 SkillRating 等条目。这样的聚合类型用 C 语言声明如下：

```
struct{char     Name[25];
       int      Age;
       float    SkillRating; }
       Employee;
```

上述声明意思是：变量Employee指向一个结构（即缩写的struct），这个结构有3个构成元素：Name（包含25个字符的字符串）、Age和SkillRating（见图6-6）。一旦声明了一个这样的集合体，程序开发人员就可以使用这个结构的名字（Employee）来指向整个集合体，或者用结构的名字跟一个圆点和字段名（如Employee.Age）来表示集合体中的单个**字段**（field）。

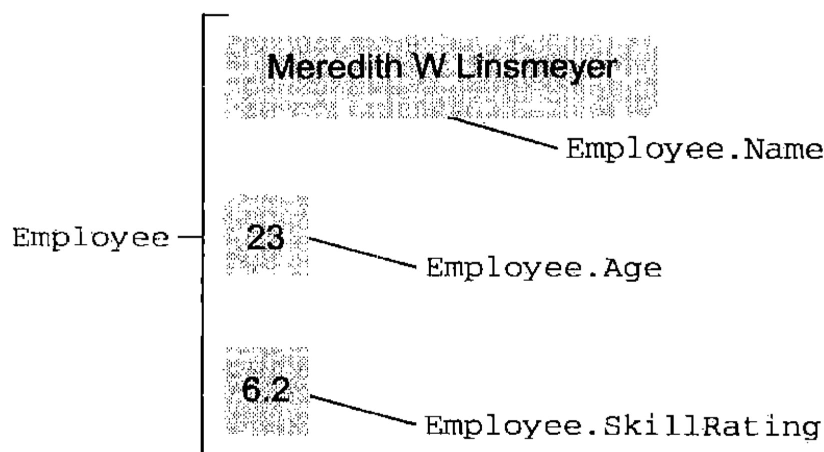


图6-6　异构数组Employee的概念结构

在第 8 章，我们将会看到诸如数组这样的概念结构是如何在计算机内部真正实现的。特别是，我们将会学到，一个数组里面的数据可以散布在主存储器或海量存储器上的广大区域内，这就是将数据结构表达成概念上的数据形态或者数据布局的原因。当然，计算机存储系统中的实际布局也许会与概念上的布局有很大的不同。

6.2.3　常量和字面量

有时，在程序中要用到预先确定的固定值。例如，一个管理机场附近区域空中交通的程序，也许要许多次引用一些关于机场的海拔高度的数据。当编写这样一个程序的时候，在每次需要这个数据时，我们都可以以数字的形式将其引入（如 645 m）。一个值的这样一种显式出现称为**字面量**（literal）。字面量的使用导致了诸如

```
EffectiveAlt ← Altimeter + 645
```

这样的程序语句的出现，其中EffectiveAlt和Altimeter是假定的变量，而645是一个字面量。这样，赋给变量Altimeter的值加上645的结果赋给了变量EffectiveAlt。

在大多数程序设计语言中，由文字组成的字面量用引号来表述，以便与其他程序部分相区分。例如，语句

```
LastName ← "Smith"
```

可以用来把文字“Smith”分配给变量LastName，而语句

```
LastName ←  Smith
```

则是把变量Smith的值赋给变量LastName。

通常，使用字面量不是一个好的编程习惯，因为字面量会掩盖包含字面量的语句的真实意义。例如，当一个读者读到

```
EffectiveAlt ← Altimeter + 645
```

这个语句的时候，他如何知道这个645代表的是什么呢？此外，字面量的使用会使在必要时修改程序的工作变得复杂。如果将空中管制程序移植到另一个机场，那么所有对机场海拔高度的引用都将要修改。如果每一处对海拔高度的引用都使用了字面量645，那么要在整个程序中定位每一个这样的引用并且加以修改。再假设在数量上，而不仅仅是在海拔高度上，同时也使用了这个字面量645，这个问题将会变得更加复杂。我们如何能够知道哪个645需要保留，哪个需要修改呢?

为了解决这个问题，程序设计语言允许为特定的不会改变的值分配一个描述性的名字。这个名字称为**常量**（constant）。例如，在C++ 和C#语言中，声明语句

```
const int AirportAlt = 645;
```

将标识符AirportAlt与一个固定的值645（我们认为它是整型的）联系起来。在Java语言中，类似的概念表达为

```
final int AirportAlt = 645;
```

根据这些声明，描述性的名字AirportAlt能够用于字面量645出现的场合。若将这种常量用于伪代码中，语句

```
EffectiveAlt ← Altimeter + 645
```

可以改写成

```
EffectiveAlt ← Altimeter + AirportAlt
```

这种方式能够较好地表达语句的含义。此外，如果用这样的常量来替代字面量，当程序要移植到另一个海拔高度为 267 英尺的机场时，仅仅修改这个定义常量的声明语句就可以将对机场海拔高度的所有引用改为新的值。

6.2.4 赋值语句

一旦声明了用于程序的专门术语（如变量和常数），程序员就可以描述涉及的算法了。这要依靠命令语句。最基本的命令语句就是**赋值语句**（assignment statement），它将一个值赋给一个变量（更确切地说，存放在该变量所标识的存储区域中）。这样的语句的语法结构通常是由变量和一个代表赋值运算的符号以及赋值表达式组成。这种语句的语义就是通过表达式求值得到结果，从而把结果作为变量的值来存储。例如，C、C++、C#和Java语言中的语句

```
Z = X + Y;
```

是将X和Y相加的和赋给变量Z。在一些其他语言（如Ada）中，等价的语句可以写成

```
Z := X + Y;
```

注意，这些语句仅仅在赋值运算符语法表示上不同，在C、C++、C#和Java语言中，仅仅使用一个等号，而在Ada中，要用一个冒号加等号的形式来表示。也许，一个更好的赋值操作符号是APL语言中所使用的，APL是由Kenneth E. Iverson在1962年设计的。（APL是A Programming Language的缩写。）它使用一个箭头来表示赋值。因此，前面的赋值在APL语言中可以表示为

```
Z ← X + Y
```

（亦与第5章的伪代码中的一样）。

赋值语句的许多功能都与语句右边的表达式的作用域关系密切。一般而言，任何一个代数表达式都可以用在赋值表达式中，包括通常用+、–、*以及 / 符号分别代表的加、减、乘、除算术运算。一些语言把**组合用来求幂。例如，在Ada中，表达式

```
X ** 2
```

表示x^2。但是，各种语言对这种表达式的解释是不一样的。例如，对于表达式2*4+6/2，如果是从右向左求值，可以得到一个值14，而从左向右求值将得到7这个结果。这种不确定性通常是通过**运算符优先级**（operator precedence）规则来解决的，这意味着某些运算比其他运算优先。传统的代数规则指定乘和除要在加与减之前执行。根据这个惯例，前面的表达式的结果应该是11。在大多数语言中，括号比所有的运算符的优先级都高。因此，2*(4+6)/2的结果应该是10。

许多程序设计语言允许使用相同的符号表示多种类型的运算。在这些情况下，符号的意义只能根据操作数的数据类型来决定。例如，当操作数是数值时，符号+传统上表示加法。但在某些语言里，如 Java，当操作数是字符串时，该符号表示连接。也就是说，表达式

```
"abra" + "cadabra"
```

的结果是abracadabra。一个运算符的这种多种用法称为**重载**（overloading）。许多程序设计语言提供了一些常见运算符的内置重载，而另外一些程序设计语言（如Ada、C++和C#）可能允许程序设计人员定义额外的重载的意义，甚至添加额外的运算符。

6.2.5　控制语句

控制语句（control statement）是可以改变程序中语句执行次序的命令语句。在所有的程序设计结构中，某些控制语句受到了极大的关注并且引发了很大的争议。主要起因是最简单的控制语句——goto语句。它提供了一种把执行顺序转向另一个位置的手段，这个位置是用名字或数标记的，这仅仅是机器语言级的JUMP指令的直接应用。但在高级语言中，这个特点意味着程序员将写出像

```
     goto 40
20   Apply procedure Evade
     goto 70
40   if (KryptoniteLevel < LethalDose) then goto 60
     goto 20
60   Apply procedure RescueDamsel
70   ...
```

这样可读性很差的程序，而仅仅使用一个语句，如

```
if (KryptoniteLevel < LethalDose)
   then (apply procedure RescueDamsel)
   else (apply procedure Evade)
```

就可以完成相同的工作。

为了避免产生这样的复杂性，现代的程序设计语言设计出了这样的控制语句，它们使得整个的分支结构可以在一条语句中表达。选择什么样的控制语句放到一个语言中是一种设计决策。目标只是要提供一种语言，使得不仅可以以可读的形式表达算法，而且可以帮助程序员获得这

种可读性。这个目标可以这样达到，限制使用那些可以导致不良程序设计的特性，同时鼓励使用优化设计的特性。结果是称为**结构化程序设计**（structured programming）的实践，它包含了系统的组织设计方法，包含对控制语句的合理使用。这个方法的中心思想就是要设计容易理解的并且满足需求规格说明的程序。

在第5章的伪代码中，我们已经遇到两种常见的分支结构，即用if-then-else和while语句分别表示的分枝结构。这几乎出现在所有的命令型、函数式或面向对象的语言中，更准确地说，下面的伪代码语句：

```
if (condition)
     then (statementA)
     else (statementB)
```

和

```
while (condition) do
   (loop body)
```

在C、C++、C#和Java中将被写成：

```
if (condition) statementA
   else statementB;
```

和

```
while (condition)
  { loop body}
```

注意这样的事实，这些语句在4种语言中是相同的，这是因为C++、C#和Java是命令型语言C的面向对象的扩展。与之相反，在语言Ada中相应的语句将被写成：

```
IF condition THEN
  statementA;
ELSE
  statementB;
END IF
```

和

```
WHILE condition LOOP
  loop body
END LOOP;
```

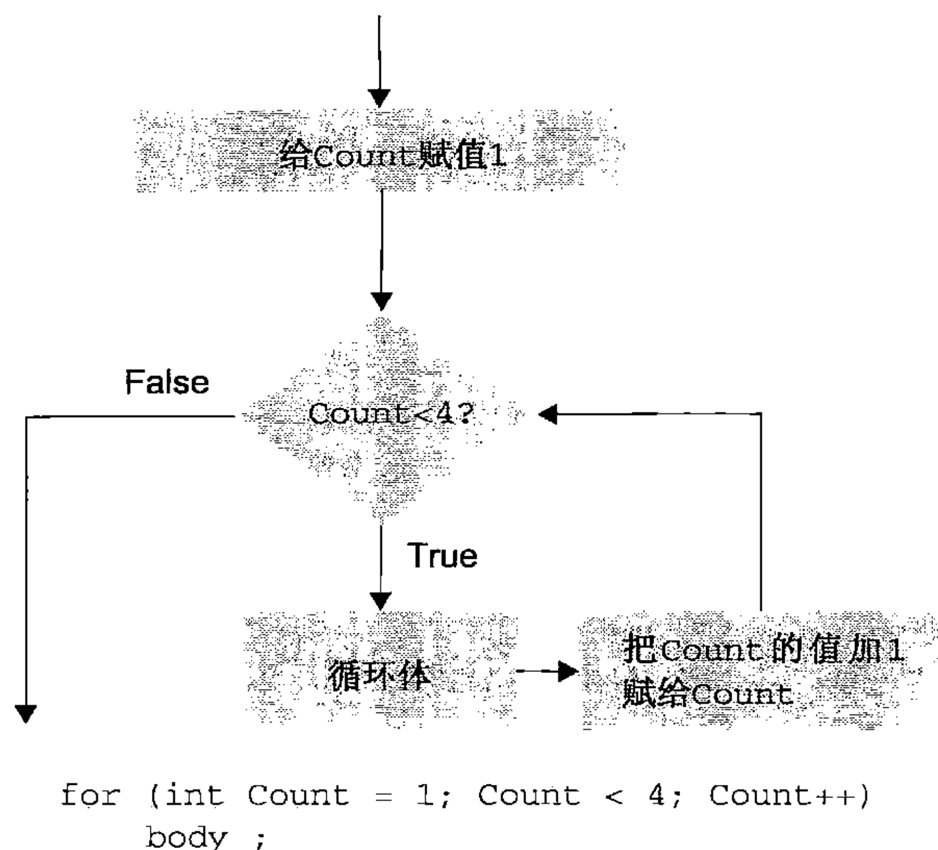
另一个常见的分支结构常用switch或case语句表示。它提供了依据赋给指定变量的值，在多个选项中选择一个语句序列的方法。例如，语句：

```
switch (variable)  {
  case 'A': statementA; break;
  case 'B': statementB; break;
  case 'C': statementC; break;
  default:  statementD}
```

在C、C++、C#和Java语言中，*statementA*、*statementB*或*statementC*语句的执行要取决于是否当前variable的值分别是A、B或C。如果variable的值是其他的值，那将执行*statementD*。在Ada中，相同的结构将被写成：

```
CASE variable IS
  WHEN 'A' => statementA;
  WHEN 'B' => statementB;
  WHEN 'C' => statementC;
  WHEN OTHERS => statementD;
END CASE
```

另外还有一个称为for结构的常见控制结构（如图6-7所示，这是C++、C#和Java语言中的表示）。这个循环结构与伪代码中的while语句相似，不同之处在于循环的所有初始化、修改和终止都在一个语句中进行。当循环体对于指定范围内的每个值都要执行一次时，这样的语句就很方便。特别地，图6-7中的语句指示循环体被重复执行，第一次Count的值为1，第二次Count的值为2，第三次Count的值为3。




```
for (int Count = 1; Count < 4; Count++)
    body ;
```

图6-7　C++、C#和Java语言中的for循环结构及其表示

程序设计语言文化

就像自然语言一样，不同程序设计语言的使用者往往会开创出不同的文化，并且经常站在各自的观点上争论语言之间的优劣。有时候这些区别是显著的，如涉及不同的程序设计范型时，而其他情况中，这种差异却很微妙。比如，尽管过程和函数在字面上存在差异（6.3节），但是C语言程序员把这两者都称作函数。这是因为在一个C程序中，过程被看做是没有返回值的函数。一个类似的例子就是C++程序员把包含在对象内的过程称为成员函数，而通常术语上我们称之为方法。造成这种差异的原因在于C++是在C语言的基础上扩展而来。另外一个文化差异出现在编写Ada程序的时候，保留字通常被加粗或使用大写字母，而这种习惯对于C、C++、FORTRAN和Java的使用者来说却不常见。

尽管本书在程序设计语言方面保持中立，并且使用通用的术语，但是每一个特定例子都采用与涉及的程序设计语言相适应的风格给出。当你碰见这些例子的时候，一定要记住的是，这些例子代表的实际上是程序设计语言中一种通用的思想，而不是作为传授某种程序设计语言细节的一种手段。不要只见树木不见森林。

根据所引用的例子，我们可以得到这样一个结论，即通用的分支结构存在于所有命令型程序设计语言和面向对象程序设计语言中，而且仅有细微的变化。从计算机科学的理论中我们可以了解到一个有些令人吃惊的结论，这就是仅仅需要这些结构中的一小部分就足以保证程序设计语言解决所有可以由算法解决的问题。我们将会在第12章研究这个问题。现在，我们只是指出，学习程序设计语言不是一个无休止的学习各种控制语句的过程，在当前的程序设计语言中可以找到的大多数控制结构本质上都是这里介绍的这些结构的变体。

Visual Basic

Visual Basic是微软公司开发的一种面向对象程序设计语言，通过它，Windows操作系统的用户可以开发他们自己的GUI应用程序。实际上，Visual Basic不止是一种语言，它还是一个完整的软件开发包，它允许程序员利用预先定义的组件（如按钮、复选框、文本框以及滚动条等）构建应用程序，并且程序员可以通过描述组件如何响应不同事件来定制这些组件。例如，对于按钮而言，程序员可以定义单击按钮时会发生什么事件。在第7章，我们将会学习到，这种通过预先定义的组件构建软件的策略是当今软件开发技术的发展趋势。

Windows操作系统的流行与Visual Basic开发包的便利性使得Visual Basic成为了当今被广泛使用的程序设计语言。如今微软公司又开发出了C#，这种优势是否能够继续我们将拭目以待。

6.2.6 注释

不管一种程序设计语言设计得多么好，也不管一个程序把该语言的特性应用得多么出色，当人们试图理解这个程序时，程序附加的信息通常能起帮助作用，或者是必须要有的。因此，程序设计语言提供了在程序中插入解释性语句方法，这些语句就是**注释**（comment）。翻译器是忽略注释语句的，因此从计算机的角度来说，注释的存在与否都不影响程序的执行。无论源程序有没有注释，对于翻译器生成的程序机器语言版本是没有影响的，但从人的角度来看，这些注释是程序的重要组成部分。若没有这些注释，对于大的复杂的程序，程序员对程序的理解肯定会受到很大的影响。

在程序中加入注释的方法通常有两种。一种是用两个特殊的记号将整个注释括起来，一个在注释的起始位置，一个在注释的尾部。另一种是标示出注释的起始位置，标记符号右边的字符全部都属于注释。在C++、C#和Java中，我们可以同时看到这两种注释方法的应用。它们用记号/*和*/把注释括起来，或者用记号//开始一个注释直至行末。因此，

```
/* This is a comment. */
```

和

```
// This is a comment.
```

都是合法的注释语句。

通常，注释要求用词少，而且含义明确。当为了制作内部文档而要求一些初级程序员使用注释语句的时候，他们容易为

```
ApproachAngle = SlipAngle + HyperSpaceIncline;
```

这样的语句写出类似“把SlipAngle和HyperSpaceIncline相加得到ApproachAngle的值”这样的注释。这样的冗余的话增加了程序的长度，但却没有解释程序。记住，注释的目的就是解释程序，而不是重复。对于这条语句的一个更合适的注释应该是解释为什么要计算ApproachAngle（如果这一点不很明显的话）。例如，注释“ApproachAngle将会在计算ForceFieldJettison-Velocity时使用，并且在此后就不再使用了”就比前面的注释更有用一些。

此外，分散在程序之中的注释有时会影响人们跟踪程序流程的能力，因此使理解程序变得比没有注释的时候还要困难。一个好方法就是将关于某个单一程序单元的注释统一放在一个位置上，比如放在该程序单元的开始位置。这就给读者提供了程序单元注释的确切地点，同时也提供了可以用来描述此程序单元的目的和综合特性的地点。如果这个格式在所有的程序单元中都采用了，写出来的程序就能在某种程度上达到一致性——每个程序单元都包括一组解释性的

语句，以及随后对该程序单元的正式表示。程序中的这种一致性提高了程序的可读性。

问题与练习

1. 为什么使用常量比使用字面量的程序设计风格更好？
2. 声明语句和命令语句的区别是什么？
3. 列举一些常用的数据类型。
4. 给出命令型程序设计语言和面向对象程序设计语言里的一些通用控制结构。
5. 数组和异构数组之间的区别是什么？

6.3 过程单元

在前面的章节中，我们已经看到了将大程序拆分成小的可管理的单元的一些好处。在本节中，我们将主要讨论过程这个概念，过程是一个命令型语言获得程序的模块化描述的主要技术。而且，在面向对象语言中，过程也是程序员指定对象如何响应外部激励的工具。

6.3.1 过程

从一般的意义上来说，**过程**（procedure）就是实现一个任务的一组指令的集合，它能够作为其他程序单元使用的抽象工具。当请求了过程提供的服务时，程序的控制权就转移给了过程，在过程执行完之后，程序控制权返回到最初的程序单元（图6-8）。将控制权转移给过程的步骤经常称为**调用**（call或者invoke）。我们将一个请求过程执行的程序单元称为调用单元。

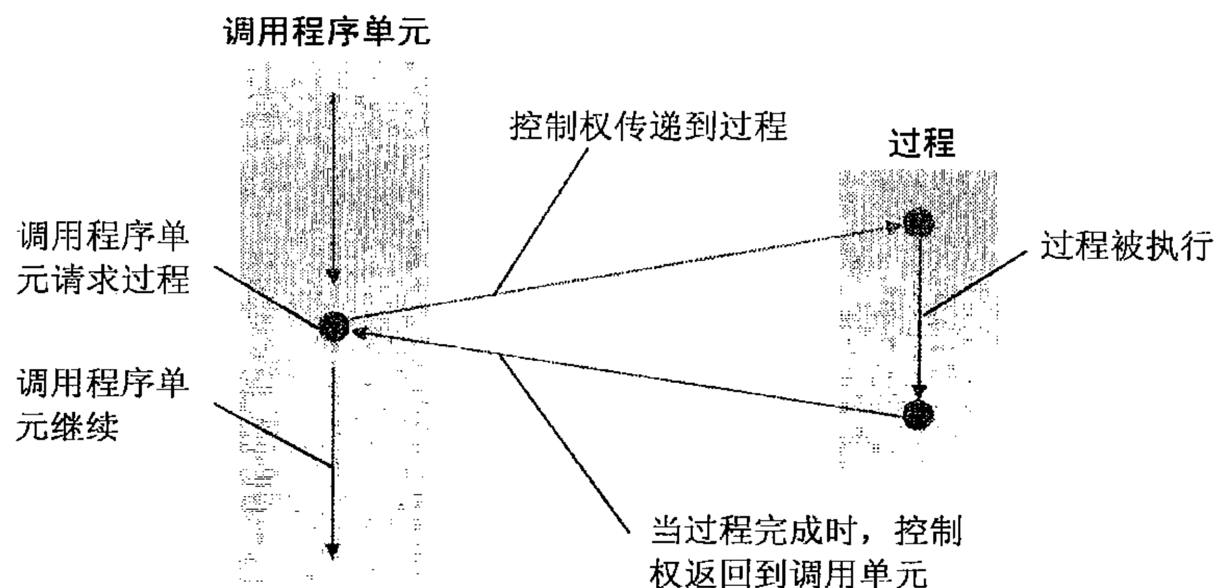


图6-8 包含一个过程的控制流

在第5章的伪代码中，过程通常以独立程序单元的形式来编写，单元以一个称为**过程头**（procedure's header）的语句开始，它标识了（在其他事情中间）过程的名称。过程头后面是定义过程细节的语句。这些语句往往以与传统的命令程序相同的方式排列，以声明语句开始（说明了过程中使用的变量），接着是命令语句（这些命令语句描述了过程执行时要履行的步骤）。

一般来说，在过程中声明的变量称为**局部变量**（local variable），意味着它只能在这个过程的内部使用。如果两个独立的过程都使用同一个变量，这很可能产生一定的混乱，而局部变量可以减少这样的冲突。[一个程序中可以引用某个变量的部分称为该变量的**作用域**（scope）。因此，局部变量的作用域就是声明它的过程。没有限制在程序中某个特定部分使用的变量称为**全局变量**（global variable），它们可以在程序的任何地方使用。大多数程序设计语言提供了声明局部变量和全局变量的方法。]

第 5 章中的伪代码使用了诸如“应用过程 DeactiveKrypton”这样的语句来请求过程的执行，但现在大多数的程序设计语言允许只通过写出过程名来调用过程。例如，如果 GetNames、SortNames 和 WriteNames 都是过程的名字，它们的功能分别是获得名字的列表、将列表排序以及打印这个列表，那么获取、排序及打印该列表的程序可以写成：

```
GetNames;
SortNames;
WriteNames;
```

而不是：

```
应用过程GetNames.
应用过程SortNames.
应用过程WriteNames.
```

注意，通过为每一个过程指定一个可以描述过程的功能的名字，这种简明扼要的形式看起来就像是反映该程序含义的命令序列。

6.3.2 参数

过程通常会使用一些通用项，这些项只有在过程被执行的时候才可以确定下来。例如，第 5 章中的图 5-11 给出了一个列表排序过程的伪代码，其中的列表不是某个特定的列表，而是一个通用的列表。在伪代码中，我们决定在过程头的括号中标识出这些通用项。因此，在图 5-11 中，过程以

procedure Sort(List)

开始，并在接下来使用List指代需要排序的列表，从而进一步描述列表排序过程。如果要应用这个过程来为一个参加婚礼的客人列表排序，我们仅仅需要假设通用项List指代参加婚礼的客人列表。如果将要排序一个会员列表，我们只要将通用项List解释成该会员列表。

过程内部的这些通用项称作**参数**（parameter）。更准确地说，这些在过程内部使用的项称为**形参**（formal parameter），并且当过程被调用的时候，赋给形参的值称为**实参**（actual parameter）。在某种程度上，形参就像是过程体上的槽口，而当过程被调用的时候，实参就好似被塞入了这个槽口中。

就伪代码来说，大多数编程语言要求在定义一个过程时将形参列在过程头的括号里。例如，图6-9给出了用C语言编写的名字是ProjectPopulation的过程的定义。该过程期望在它被调用的时候，接受一个确定的年增长率值。在这个增长率的基础上，假设初始数量为100，过程计算出未来10年中某个种群的数量，并且将结果存储在称为Population的全局数组中。

大多数程序设计语言在调用过程的时候也使用括号来标识实参。也就是说，调用过程的语句要包括过程的名字并在紧接名字的括号中给出实参的列表。因此，像

```
使用增长率0.03应用过程ProjectPopulation
```

这样的伪代码语句可以用C语言语句

```
ProjectPopulation(0.03);
```

表示，它是用增长率的值0.03来调用图6-9中的过程ProjectPopulation。

当过程不只包含一个参数的时候，实参要与过程头的形参序列一一对应——第一个实参对应第一个形参，依此类推。然后，实参的值就可以有效地传递给它们相对应的那个形参，从而过程得以执行。

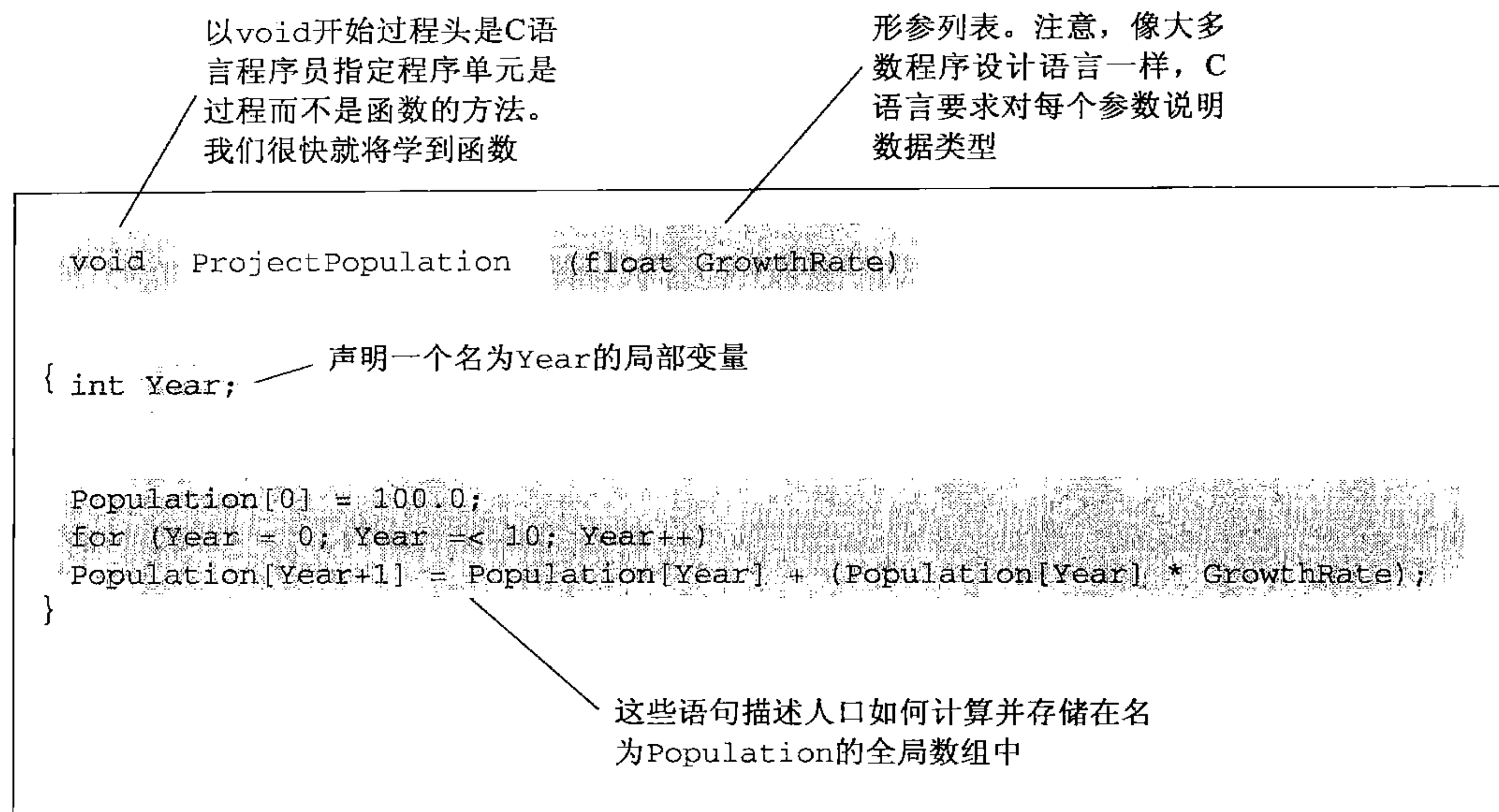


图6-9 用C语言编写的过程ProjectPopulation

为了强调这一点，假设过程PrintCheck使用这样的过程头来定义：

```
procedure PrintCheck(Payee, Amount)
```

其中Payee和Amount是过程的形参，它们分别指代了将支票支付给哪个人以及支票的数额。那么，用语句

```
PrintCheck("John Doe",150)
```

调用过程，将会使得形参Payee与实参John Doe对应，形参Amount与实参150对应，从而过程得以执行。但是使用语句

```
PrintCheck(150, "John Doe" )
```

调用过程将会使得值150赋给形参Payee，而John Doe赋给形参Amount，而这必定会导致错误的结果。

对于形参和实参之间的数据传递，不同的程序设计语言有不同的处理方法。在某些语言中，对于实参所表示的数据会产生一个副本并传给过程。使用这种方法，过程对数据的任何修改仅仅是对副本的修改——调用程序单元中的数据并没有被修改，我们称之为**按值传递**（pass by value)。注意，按值传递参数可以保护调用单元中的数据，使之不会被设计有问题的过程错误地修改。例如，如果调用单元传递一个雇员的名字给一个过程，我们当然希望过程不要改变这个名字。

但是，当参数表示一个很大的数据块时，按值传递参数效率不高。一个更高效地给过程传递参数的方法，就是在调用程序单元中告诉过程它所需的实参的地址，从而使过程可以对实参进行直接存取。对于这种方法，我们称为**按引用传递**（pass by reference)。注意，按引用传递参数允许程序修改调用单元中的数据。这个方法对于为列表进行排序的过程来说是很有用的，因为调用这样的过程目的就是改变列表。

例如，假设一个过程 Demo 定义为：

```
Procedure Demo(Formal)
Formal←Formal + 1;
```

此外，假设变量Actual被赋予一个值5，我们用下面的语句调用Demo：

```
Demo(Actual)
```

然后，如果参数是按值传递的，在过程中对Formal的改变不会影响变量Actual的值（参见图6-10）。但是，如果是按引用传递的话，那么Actual的值将会增加1（参见图6-11）。

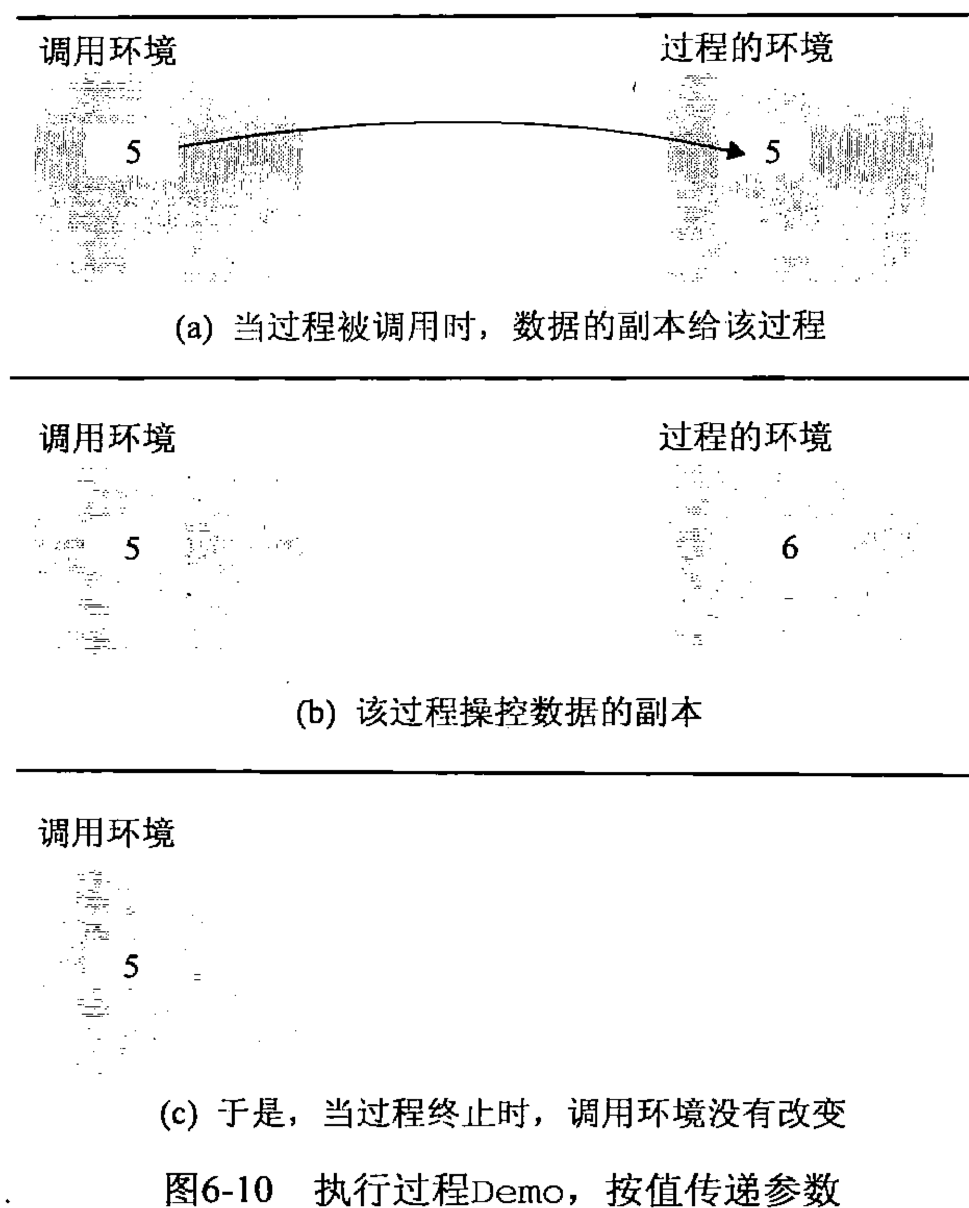


图6-10 执行过程Demo，按值传递参数

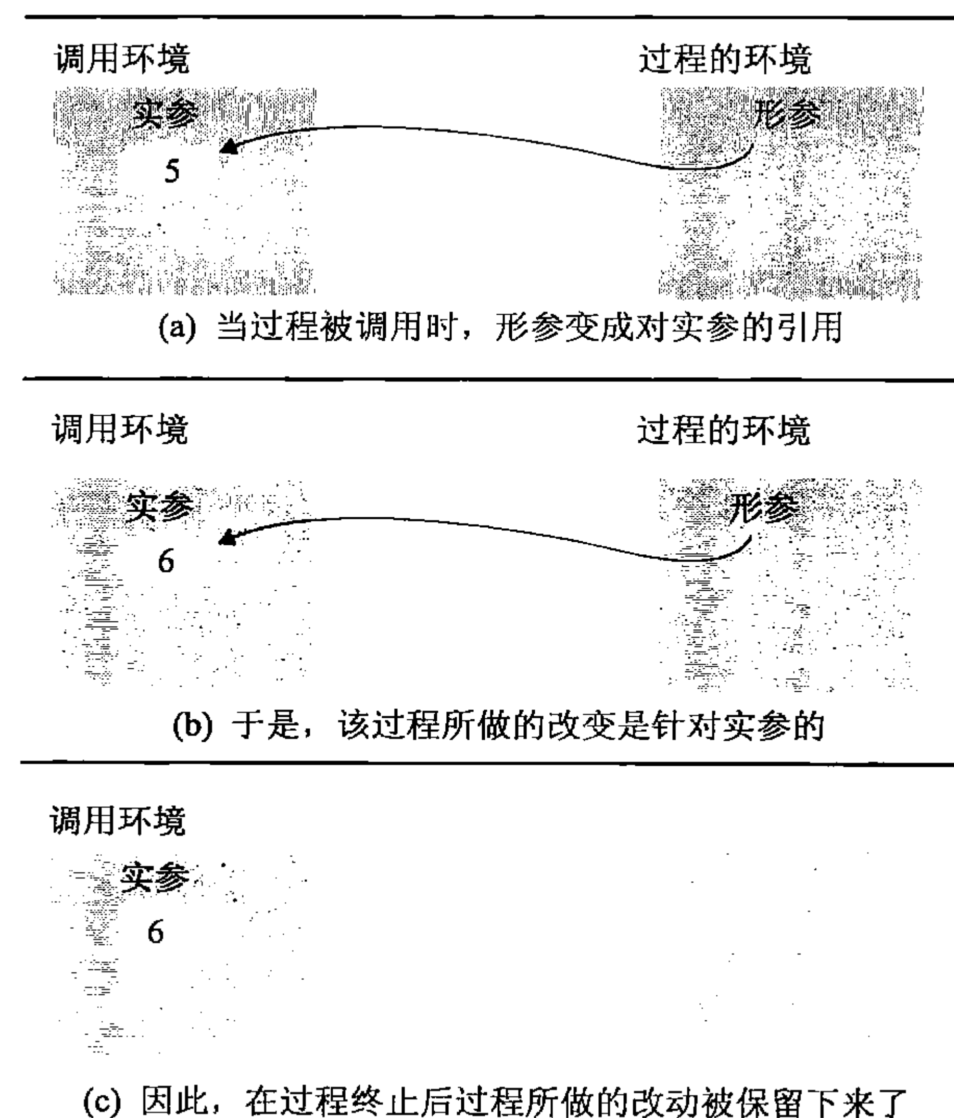


图6-11 执行过程Demo，按引用传递参数

不同的程序设计语言提供了不同的参数传递技术，但是在任何情况下，参数的使用都允许过程以通用的意义书写，并在适当的时候应用于特定的数据。

6.3.3 函数

让我们暂停一下来考虑过程概念的一个微小的变化，该变化存在于许多程序设计语言中。有时，过程的目的是要产生一个值，而不是完成一个动作。（考虑这样两个过程之间的差别，一个过程是估计售出的小商品的数量，另一个过程用来玩一个小游戏，前者重点是为了产生一个值，而后者是为了完成一个动作。）如果目的是产生一个值，那么这个“过程”是作为一个函数来执行的。这里，**函数**（function）是指一个类似于过程的程序单元，但它把一个值作为“该函数的值”传递给调用程序单元。也就是说，函数的执行就是计算出一个值并且将这个值送回到调用程序单元中。这个值可以存储在一个变量里为以后使用，也可以立即用于计算。例如，C、C++、Java或者C#的程序员可以编写

```
ProjectedJanSales = EstimatedSales(January);
```

来把调用函数`EstimatedSales`产生的结果（一月份共售出了多少小商品）赋值给变量`ProjectedJanSales`。或者，程序员可以编写

```
if (LastJanSales < EstimatedSales(January))...
   else ...
```

依据今年一月份的销售是否好于去年同期来产生不同的动作。注意，在第二种情况中，由函数计算出来的值用来决定执行哪个分支，但是不会被存储起来。

在程序中定义函数的方式与定义过程的方式基本相同。它们的不同仅仅在于，函数头通常以指定返回值的数据类型开始，以返回语句来结束函数定义——这个返回语句明确了返回值。图6-12给出了函数`CylinderVolume`的C语言定义。（实际上，一个真正的C程序员会使用一种更简洁的方式，我们之所以使用这种详细的样式是为了教学的需要。）当函数被调用的时候，函数的形参`Radius`和`Height`接受确定的值，并且函数使用这些参数返回经过计算得到的圆柱体积。因此，在程序的其他地方，类似于下面的语句：

```
Cost = CostPerVolUnit * CylinderVolume(3.45, 12.7);
```

可用来调用这个函数，以求出一个半径为3.45，高为12.7的圆柱的费用。

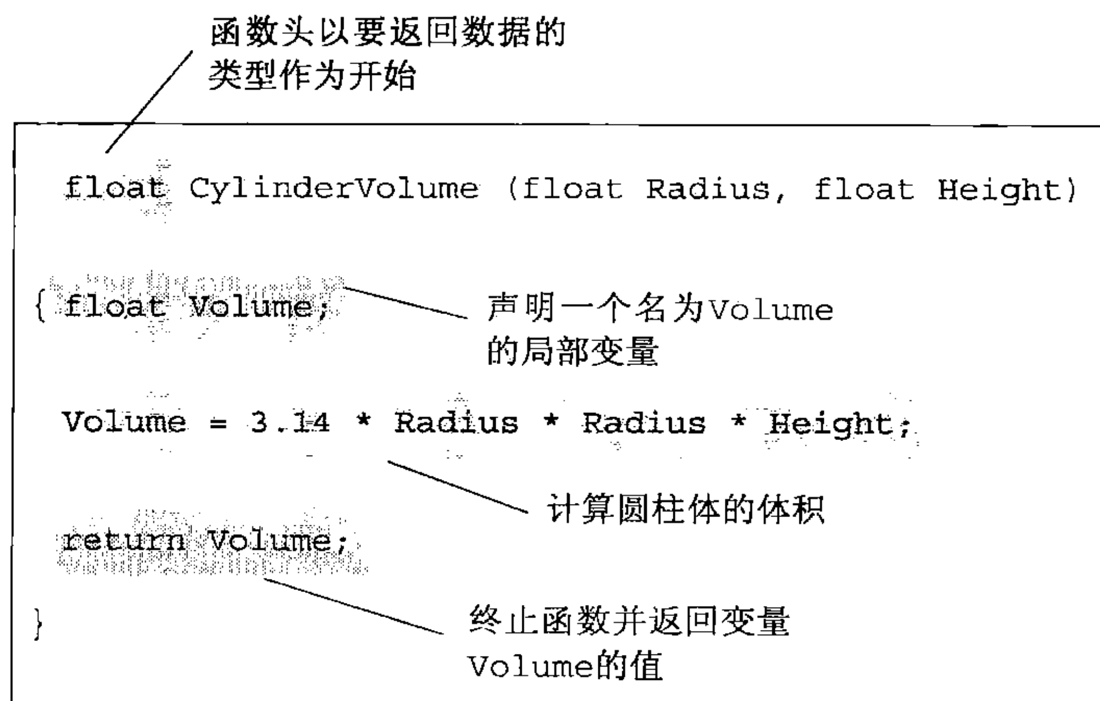


图6-12　用C语言编写的函数CylinderVolume

事件驱动软件系统

文中，在我们已经考虑过的情况中，过程的激活都是作为程序中其他位置的语句明确地调用过程的结果。此外，还有一些情况是这样的，过程是由一个事件的发生激活的，例如在GUI中，有种过程描述了当一个按钮被单击的时候应该产生什么动作，这种过程不是由其他程序单元调用激活的，而是作为单击按钮这一事件的结果激活的。这种过程是通过事件而不是明确的请求来激活的软件系统称作**事件驱动**（event-driven）系统。简言之，一个事件驱动软件系统是由这样的过程组成，它们描述各种事件发生时应该做什么。当系统执行时，这些过程等待，直到与它们对应的事件发生，然后它们被激活，并在完成它们的任务后回到等待状态。

问题与练习

1. 变量作用域是什么？
2. 过程和函数的区别是什么？
3. 为什么许多程序设计语言执行I/O操作的方式很像是调用过程？
4. 形参和实参的区别是什么？
5. 当用一个现代程序设计语言写程序时，程序员都倾向于使用动词来命名过程，使用名词来命名函数，为什么？

6.4 语言实现

在本节中，我们将研究把高级语言编写的程序转换为机器可执行形式的过程。

6.4.1 翻译过程

将一个程序从一种语言转换为另一种语言的过程称为**翻译**（translation）。原始形式的程序称作**源程序**（source program），翻译后的版本称作**目标程序**（object program）。翻译过程包括3部分工作，分别是词法分析、语法分析和代码生成，实现相应行为的单元分别称为**词法分析器**（lexical analyzer）、**语法分析器**（parser），以及**代码生成器**（code generator），参见图6-13。

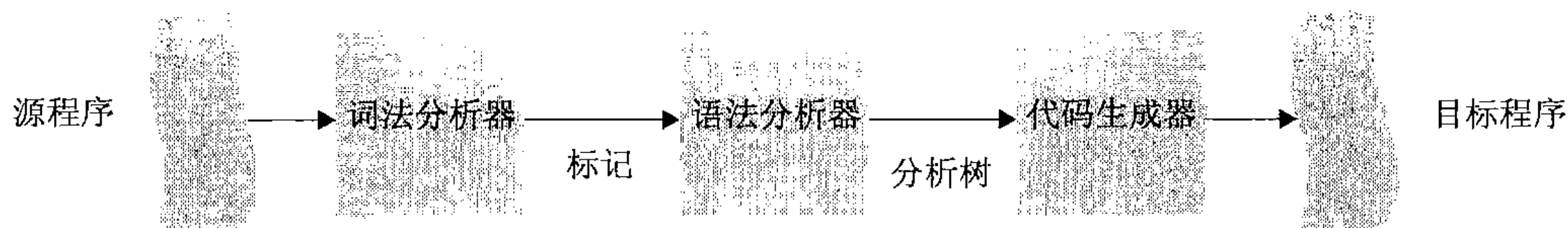


图6-13 翻译过程

词法分析是识别源程序中构成单个实体——**标记**（token）——的符号串的过程。例如，3个符号的153不应该解释成一个1、一个5和一个3，而是应该识别为一个数值。同样，程序中的一个单词，尽管由独立的符号组成，也应该解释成一个单元。大多数人进行词法分析都是下意识的。当要求大声朗读的时候，我们都是读出一个词来，而不是逐个读出单个字母。

因此，词法分析器逐个符号地读源程序，并且识别出哪些符号的组合可以代表一个标记，并且根据它们是否是数、词、算术运算符等将这些标记分类。词法分析器对标记及其分类进行编码，并将它们提交给语法分析器。在此过程中，词法分析器跳过了所有的注释语句。

因此，语法分析器将程序看做是由词法单元（标记）组成的，而不是由独立符号组成的。语法分析器的工作就是将这些单元组合成语句。实际上，语法分析是标识程序中语法结构和辨认每个成分作用的过程。正是语法分析技术使得人们在读句子

The man the horse that won the race threw was not hurt.

时，会停顿一下。（试一试这句话："That that is is . That that is not is not. That that is not is not that that is."！）

为了简化语法分析过程，早期程序设计语言坚持每个程序的语句都要以一种特定的方式定位在打印页上。这种语言称为**固定格式语言**（fixed-format language）。今天，大多数程序设计语言都是**自由格式语言**（free-format language），意味着不再苛求语句的位置安排了。从人的角度来看，自由格式语言的好处在于程序员可以编写可读性更高的程序。在这种情况下，通常使用缩进来帮助读者更好地把握语句的结构。对于一个程序员，不应该写

```
if Cost < CashOnHand then pay with cash else use
credit card
```

而应该写

```
if Cost < CashOnHand
  then pay with cash
  else use credit card
```

Python

Python是Guido van Rossum于20世纪80年代后期创建的程序设计语言。今天，Python被广泛应用于Web应用开发、科学计算，并用作面向学生的入门语言。Python强调可读性，包含命令型编程范型、面向对象型编程范型及函数式编程范型。Python还是现代语言中的一种固定格式语言。它使用缩进来表示程序块，而非标点符号或保留字。

对于一台计算机，为了分析以自由格式语言编写的程序，程序设计语言的语法必须设计用来帮助识别程序的结构，而不管源程序使用了多少空格。为了达到这个目的，大多数自由格式语言都使用诸如分号这样的标点符号来表示语句的结束，另外还使用诸如 `if`、`then` 和 `else` 这样的**关键字**（key word）来表示单个短语的开始。这些关键字通常都是**保留字**（reserved word），就是说程序员在程序中不能把它们用于其他目的。

Java和C#的实现

在某些情况下，如控制动画网页时，软件必须经过因特网传送，且在远程机器上执行。如果软件是以源程序形式提供的，那在目的地将产生额外的延迟，这是因为软件在执行前要被翻译成合适的机器语言。但是，以机器语言的形式提供软件意味着要根据远程计算机上使用的机器语言提供不同版本的软件。

通过设计能够翻译源代码的"通用机器语言"（在Java中称为字节编码，在C#中称为.NET通用中间语言），Sun Microsystems和微软解决了此问题。虽然这些语言不是真实的机器语言，但它们被设计成了快速可翻译的语言。这样，如果用Java或C#编写的软件被翻译成了合适的"通用机器语言"，那么它能被传送到因特网中的另外一台机器上，在那里被高效地执行。在某些情况下，这个执行是由解释器完成的，而在其他一些情况下，通用机器语言在执行前被快速翻译，这个过程称为**即时编译**（just-in-time compilation）。

语法分析过程基于一系列语法规则，这些规则定义了程序设计语言的语法。总地来说，这些规则称为**文法**（grammar）。表达这些规则的一种方法是借助**语法图**（syntax diagram），它是程序文法结构的图形化表示。图 6-14 给出了第 5 章伪代码中的 if-then-else 语句的语法图。这个图说明 if-then-else 结构以关键字 if 开始，然后是一个布尔表达式，接着是关键字 then，随后是一个语句。此结构的后面有没有 else 和语句都是允许的。注意，实际出现在 if-then-else 语句中的项都使用椭圆形框，而需要进一步描述的项，诸如布尔表达式和语句等，都在矩形框中。需要进一步描述的项（矩形框中的）称为**非终结符**（nonterminal），而出现在椭圆形框中的项称为**终结符**（terminal）。在一个程序设计语言语法的完整描述中，非终结符由额外的图表描述。

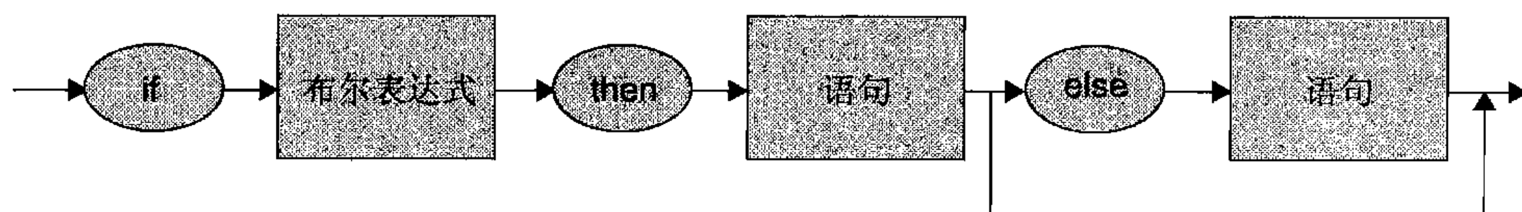


图6-14 if-then-else伪代码语句的语法图

作为一个比较完整的例子，图 6-15 给出了一组语法图，它们描述了一个称为表达式（可以是简单的数学表达式结构）的结构的语法。第一个图描述了一个表达式，这个表达式由一个项（term）组成，后面可以跟（也可以不跟）一个+号或-号，运算符后面跟有另一个表达式。第二个图描述了一个项，这个项由单个因子组成，或者由一个因子后面跟一个×号或÷号，然后再跟另一个项组成。最后一个图描述了一个因子，这个因子由 x、y、z 中的一个字符组成。

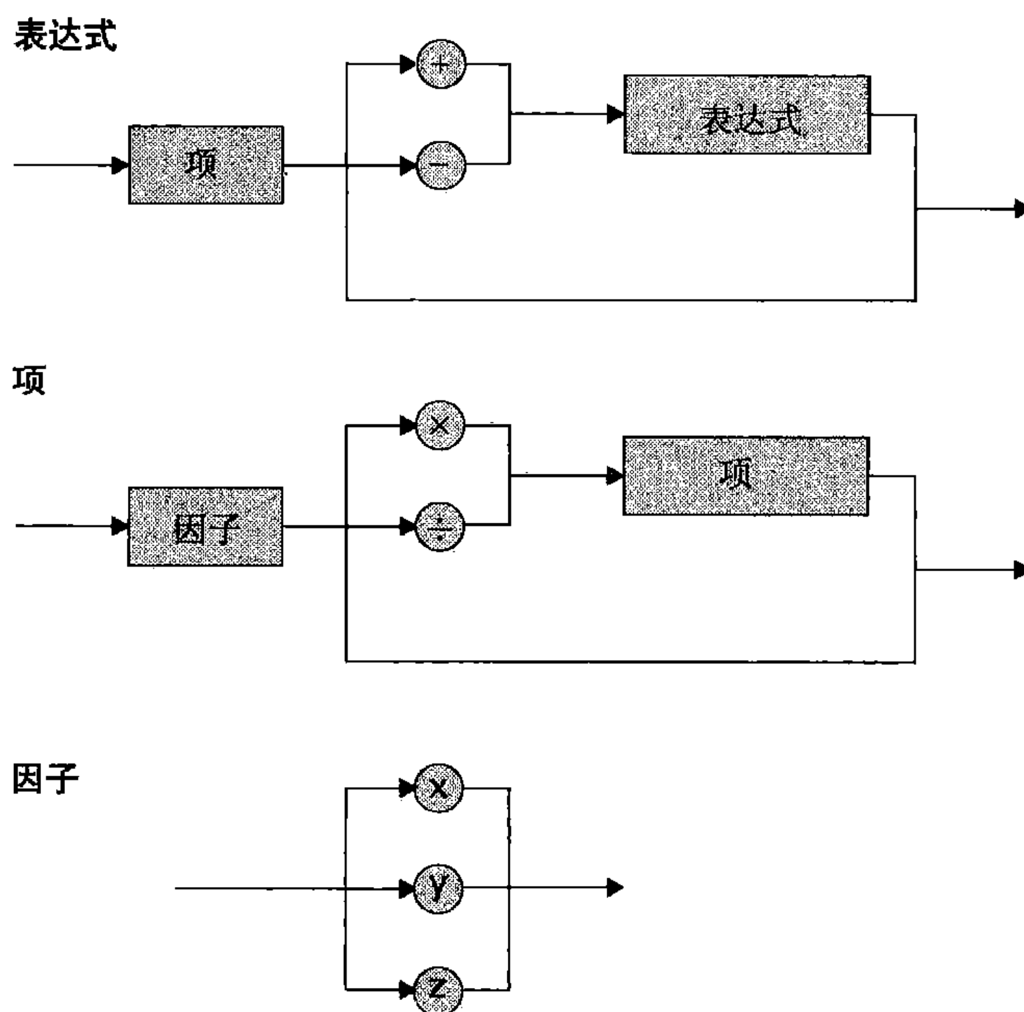


图6-15 一个简单的代数表达式的语法图

一个特定的串符合一组语法图的方式还可以用**语法分析树**（parse tree）进行图形化表示，根据图 6-15 中所示的语法图，图 6-16 描述了字符串 x+y×z 的语法分析树。注意，树的顶端以非终结符表达式开始，并且在每一层都给出了本层的非终结符是如何分解的，这个过程直到获得该串本身中的全部符号才结束。该图还特别说明（根据图 6-15 中的第一个图），一个表达式

可以分解成一个项，后面跟有+号，然后再跟一个表达式。接着，项可以分解成（用图 6-15 中的第二个图）一个因子（结果是符号×），最后的表达式可以分解（用图 6-15 中的第三个图）为一个项（结果是 y×z）。

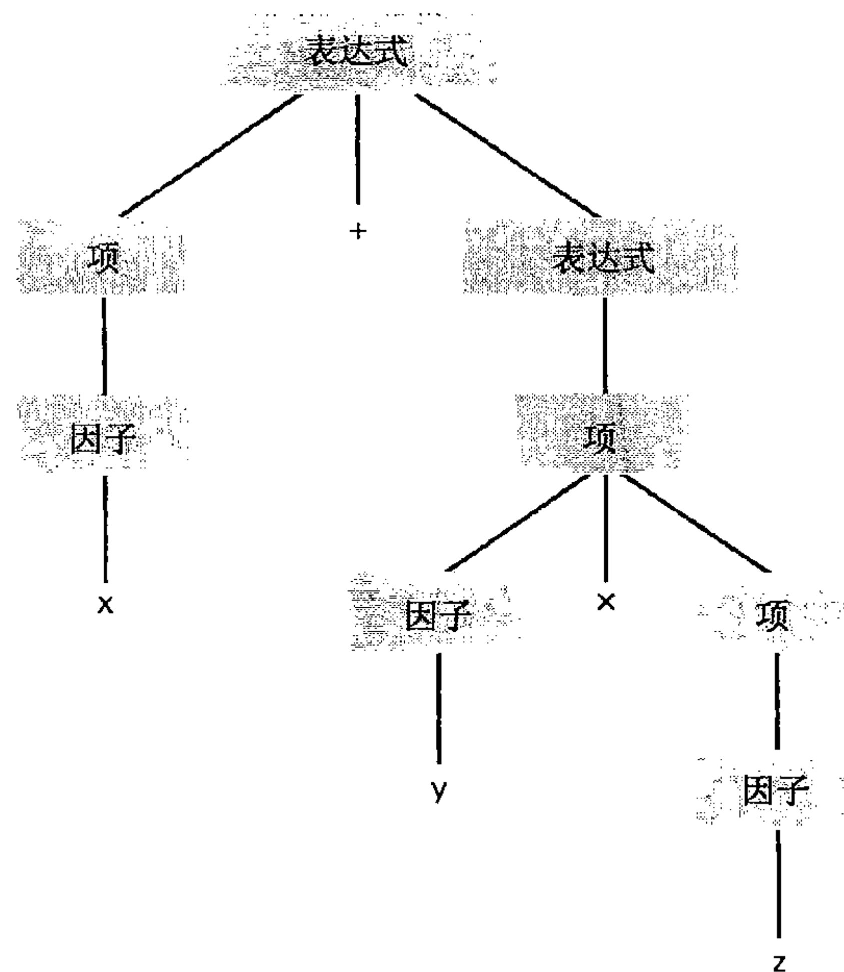


图6-16　基于图6-15的x+y×z字符串的语法分析树

语法分析过程本质上就是为源程序构建语法分析树的过程。的确，一个语法分析树代表了语法分析器对程序文法构成的理解。因此，描述程序文法结构的语法规则是不允许同一个字符串出现两个不同的语法分析树的，因为这将导致语法分析器内部发生混乱。如果一个文法允许同一个字符串有两个不同的语法分析树，我们称之为**多义文法**（ambiguous grammar）。

语法中的这种歧义是很细微的。事实上，图 6-14 中所示的规则存在这样的缺陷，对于下面的语句可以生成如图 6-17 所示的两个语法分析树：

if *B1* then if *B2* then *S1* else *S2*

注意，这两个解释是明显不同的。第一个表示语句 *S2* 在 *B1* 为假时执行，而第二个表示语句 *S2* 仅当 *B1* 为真并且 *B2* 为假时执行。

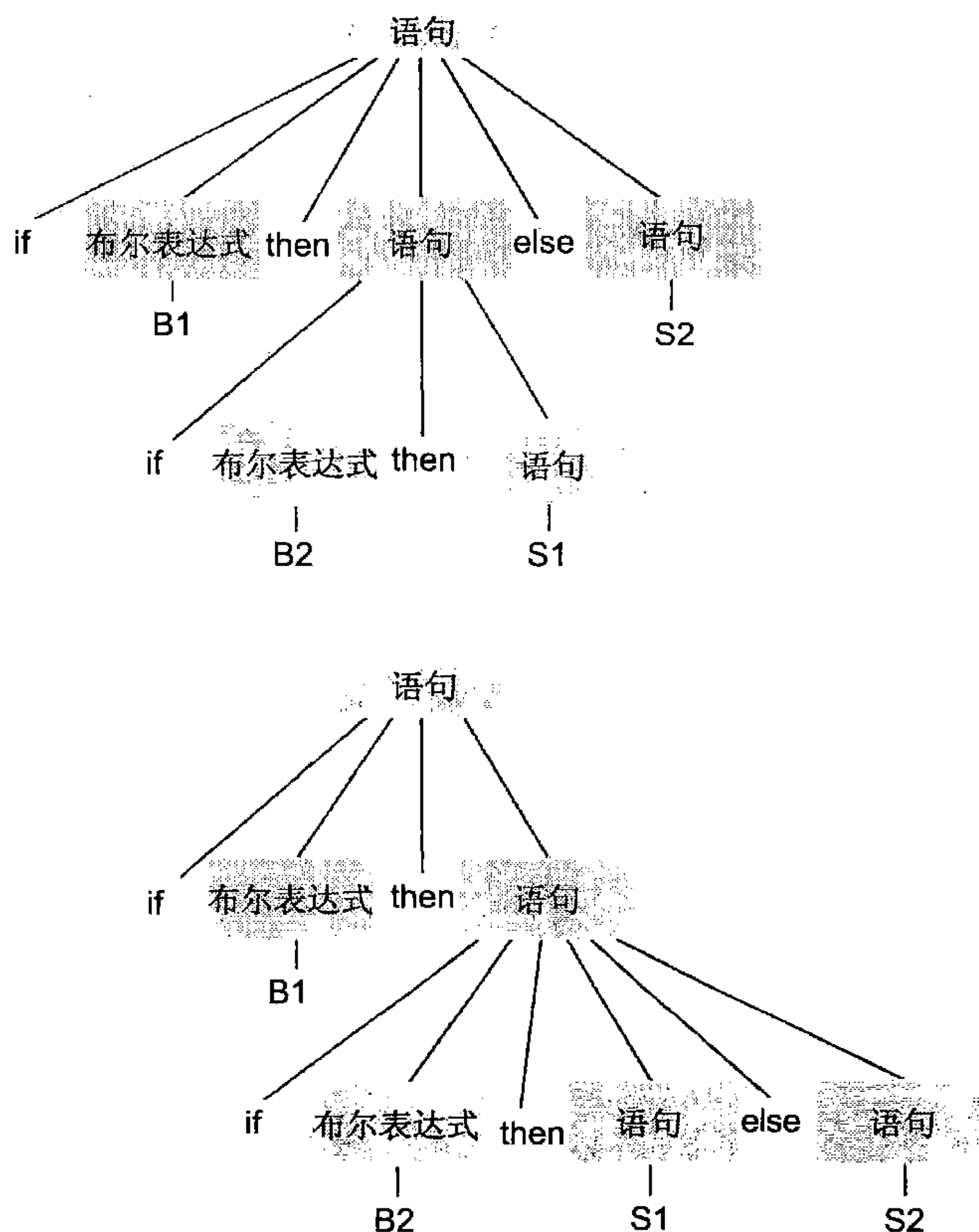


图6-17　语句if *B1* then if *B2* then *S1* else *S2* 的两个不同的语法分析树

正式的程序设计语言的语法定义要避免这样的混乱。在伪代码中，我们通过使用括号来避免这样的问题。我们可以写

```
if B1
    then (if B2 then S1)
    else S2
```

以及

```
if B1
    then (if B2 then S1
               else S2)
```

来区分这两种可能的解释。

当语法分析器分析一个程序的文法结构时，它能够标识单独的语句，并能够区分声明语句和命令语句。当识别声明语句时，它将这些声明的信息记录在一个**符号表**（symbol table）中。因此符号表中包含了诸如变量名等信息和与其对应的数据类型和数据结构的信息。然后，当分析到

```
z ← x + y;
```

这样的命令语句时，语法分析器会以这些信息为依据来进行分析。特别是，为了确定符号+的含义，语法分析器必须要知道x和y的数据类型。如果x是实型而y是字符型，将x和y相加是没有任何意义的，并且将会报告出错；如果x和y都是整型，那么语法分析器将会请求代码生成器生成相应的整数加法操作码的机器语言指令；如果x和y都是实型，那么语法分析器将会请求代码生成器生成相应的浮点数加法操作码的机器语言指令；如果都是字符类型，语法分析器将会请求代码生成器建立一串机器语言指令来完成相应的连接操作。

如果x是整型而y是实型这种特殊的情况，加法的概念是可用的，但是值不能以兼容的形式编码。在这种情况下，语法分析器可能选择使代码生成器生成指令把其中的一个值转换为另一种类型，然后再执行加法运算。这种类型的隐式转换称为**强制类型转换**（coercion）。

许多程序设计语言的设计者都反对强制类型转换，因为隐式类型转换会改变数据项的值，并因此导致微妙的程序bug。他们认为，经常需要强制类型转换就意味着程序设计语言在设计上存在纰漏，因此不应该使用语法分析器来迁就。因此，大多数现代程序设计语言都是**强类型**（strongly typed）的，这意味着一个程序请求的动作必须包含允许的数据类型，不允许强制类型转换。一些程序设计语言（如Java）支持进行强制类型转换，只要它是**类型提升**（type promotion），意思是将一个低精度值转换为一个较高精度的值。可能改变一个值的隐式强制类型转换将被报告为错误。在大多数情况下，程序设计人员仍可以请求这种类型转换，方法是进行显式强制**类型转换**（type cast）。显式强制类型转换会通知编译器：程序设计人员知道将应用类型转换。

翻译过程的最后一步就是**代码生成**（code generation），它是生成机器语言指令以实现语法分析器识别出的语句的过程。这个过程涉及许多问题，其中一个就是要生成效率高的机器语言版本的程序。例如，我们考察语句

```
x ← y + z;
w ← x + z;
```

的翻译工作。如果这些语句作为两个独立语句来进行翻译，那么在执行加法操作之前，每一条语句都需要数据从主存传送到CPU。然而，效率可以通过这样的认识获得：一旦第一条语句被执行之后，x和z的值已经存在于CPU的通用寄存器中，所以执行第二条语句的时候就不再需要

从内存中读取这两个数了。这种方法就称作**代码优化**（code optimization），它是代码生成器的一个非常重要的工作。

最后，我们应当注意的是，词法分析、语法分析和代码生成这 3 个步骤并不是严格按照顺序执行的。相反，这些步骤是交织在一起的。词法分析器首先从源程序中读取字符，并且标识出第一个标记。它将这个标记传送给语法分析器。每当语法分析器从词法分析器接收到一个标记时，就开始分析读取的文法结构。此时，它可能会向词法分析器请求另一个标记，而如果语法分析器认为已经读到了一个完整的短语或者语句，那么它就会请求代码生成器产生相应的机器指令。每一个这样的请求都会使代码生成器生成机器指令，并且将其加入到目标程序中。将一个程序从一种语言翻译成另一语言的工作很自然符合面向对象范型。源程序、词法分析器、语法分析器、代码生成器以及目标程序本身都是对象，每一个对象都在实现自己的任务的同时通过来回传递消息与其他对象进行交互（见图 6-18）。

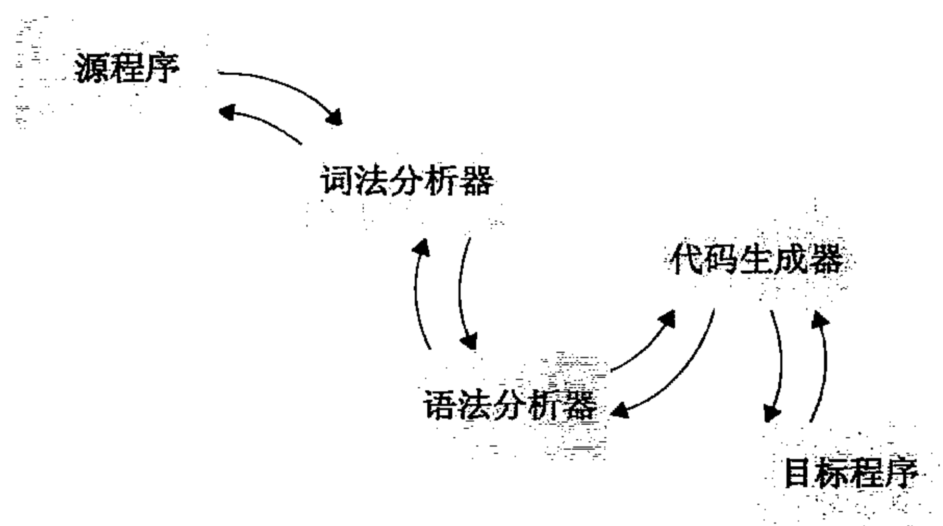


图6-18 翻译过程的面向对象方法

6.4.2 软件开发包

像编辑器和翻译器这样的在软件开发过程中应用的软件工具，通常组合成一个软件包，来实现一个集成的软件开发系统的功能。根据3.2节中的分类框架，这样的系统属于应用软件。通过使用该应用软件包，程序员可以很方便地在一个编辑器中编写程序，使用翻译器将程序转换成机器语言，并且可以使用各种各样的调试工具来跟踪出错程序的执行，以发现哪里出现了问题。

使用这样的集成系统的好处很多，最明显的就是程序员在需要修改和测试程序时，可以很容易在编辑器和调试工具之间来回倒换。此外，许多软件开发包允许开发中的相关程序单元以合适的方式链接，以便简化对相关程序单元的存取。一些软件包还维护这样的记录，即在上次基准制定以来，一组相关程序单元中哪些已经做了修改。这些功能对于许多相关程序单元是由不同程序员开发的大型软件系统开发来说是非常有用的。

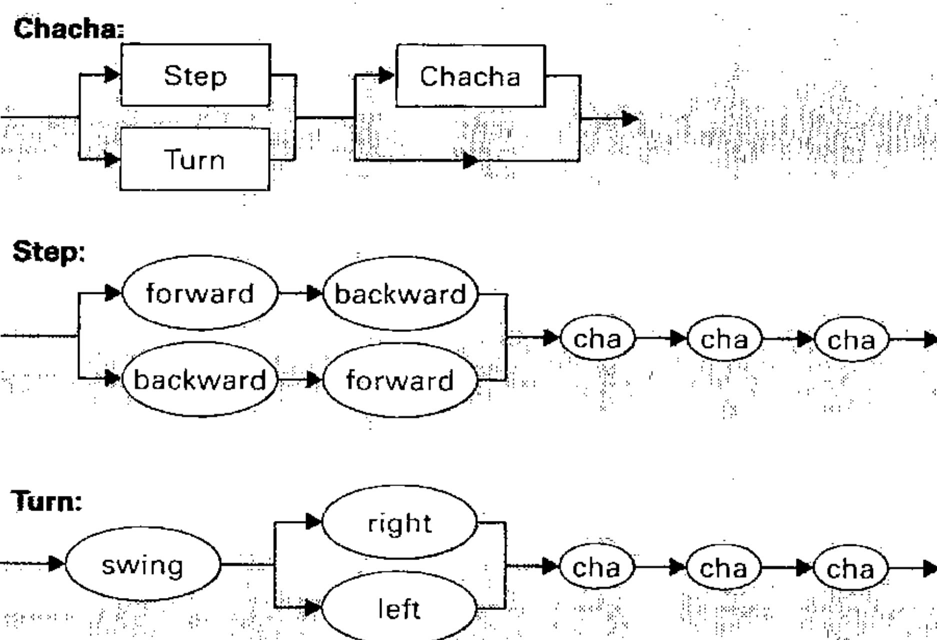
从小范围讲，软件开发包中的编辑器通常根据正在使用的程序设计语言进行定制。这样的编辑器通常提供自动行缩进功能，这已成为目标语言事实上的标准。有时，对于关键字，只要程序员键入前面少数几个字符，编辑器就能够识别并且自动补全。此外，编辑器可以突出源程序中的关键字（也许使用颜色），使程序易读。

在第 7 章中，我们将学到软件开发人员越来越多地研究这样的方法，即如何用预制的称为构件的程序块构建新的软件系统，这导致了一种新的称为构件架构的软件开发模型的产生。基于构件架构模型的软件开发包通常使用图形界面，在监视器屏幕上由图标表示各种构件。在这种环境下，程序员（或者构件装配人员）用鼠标选择所需要的构件。选好的构件可以用软件开发包的编辑器进行定制，然后用鼠标进行定位和单击就可以加到其他构件上。这种软件包代表

了在研究更好的软件开发工具的方向上前进了一大步。

问题与练习

1. 描述翻译过程的3个主要步骤。
2. 什么是符号表？
3. 终结符和非终结符的区别是什么？
4. 基于图6-15中的语法图，为表达式x× y + x + z画出语法分析树。
5. 根据下面的语法图，描述遵循语法结构Chacha的字符串。



6.5 面向对象程序设计

在 6.1 节中，我们看到面向对象程序设计范型必然需要开发称为**对象**（object）的活动程序单元，每一个对象都包含了描述对象怎样响应各种激励的过程。一个问题的面向对象解决方法就是标识出涉及的对象，并将其作为一个独立的单元来描述。接着，面向对象程序设计语言提供了描述对象及其行为的语句。在本节中，我们将引入一些以 C++、Java 和 C#语言形式出现的语句，这 3 种语言都是当今比较著名的面向对象程序设计语言。

6.5.1 类和对象

我们可以考虑开发一个简单的计算机游戏的任务，在这个游戏中，玩家要通过高能量激光器向从天上掉下来的流星进行射击来保卫地球。每个激光器都有一定的内部能量源，而每一次射击将消耗一部分能量。一旦能量用尽，激光器就失去了作用。每一个激光器都应该能响应瞄准右面一点、瞄准左面一点或发射激光束的命令。

在面向对象范型中，计算机游戏中的每一束激光都作为一个对象来实现，每个这样的对象都包含了它的剩余能量的记录以及修改目标和发射激光束的过程。既然所有的激光对象都有同样的属性，它们可以使用一个公用模板来构建。在面向对象范型中，这样的一组对象的模板称作**类**（class）。

在第 8 章我们将探究类和数据类型之间的相似点。现在，我们只需注意类描述的是一组对象的共同特征，这与基本数据类型整型的概念包含像数字 1、5 和 82 这样的数的一般特征类似。一旦程序员在程序里面包含一个类的描述，那个模板就可以用来构建和操作那一“类型”的对象，这相当于基本的整型允许操作整型的“对象”。

在C++、Java和C#语言中，类使用下列形式的语句来描述：

```
class Name
{
      .
      .
      .
}
```

其中，*Name*是一个名字，在程序的其他地方可通过这个名字来引用该类。括号中的是被描述的类的属性。图6-19特别展示了描述计算机游戏中激光结构的名为LaserClass的类。这个类包含1个名字为RemainingPower的整型变量以及3个名字分别为turnRight、turnLeft和fire的过程的声明，这些过程描述了完成相应动作的执行步骤。因此，任何一个从该模板构建的对象都包含如下特性：1个名字为RemainingPower的变量以及3个名字分别为turnRight、turnLeft和fire的过程。

```
class LaserClass
{ int RemainingPower = 100;
  void turnRight ( )
  { ... }
  void turnLeft ( )
  { ... }
  void fire ( )
  { ... }
}
```

保留在每个该类型对象里的数据的描述

描述该类型对象应该如何应对各种消息的方法

图6-19　描述计算机游戏中一种激光武器的类结构

对象内部的变量，例如RemainingPower，称为**实例变量**（instance variable），而在对象里的过程称为**方法**（method，对于C++来说称作成员函数）。注意，在图6-19中，实例变量RemainingPower使用类似于在6.2节中的声明语句来描述，而方法使用6.3节中的函数或者过程的方式来描述。实例变量的声明和方法的描述是最基本的命令型程序设计的概念。

一旦在游戏程序中描述了类LaserClass，我们就可以声明3个LaserClass“类型”的变量Laser1、Laser2、Laser3，这是通过语句

```
LaserClass  Laser1, Laser2, Laser3;
```

来实现的。注意，这与我们在6.2节中所学的声明3个整型变量x、y和z的语句

```
int  x, y, z;
```

的格式是一样的。它们都包括了一个类型名，以及出现在类型名后的将要声明的变量列表。二者都由变量名后面跟着将要声明的一系列变量组成。区别在于，后者所说的变量x、y和z在程序中用来指向一个整型项（基本类型），而前者所说的变量Laser1、Laser2和Laser3在程序中用来指向一个LaserClass“类型”的项（这是在程序中自己定义的“类型”）。

一旦我们声明了LaserClass“类型”的变量Laser1、Laser2和Laser3，就可以给它们赋值。在这种情况下，所赋的值必须是与LaserClass类型相一致的对象。这些赋值可以通过赋值语句来进行，但是在声明变量时，在同一个声明语句内给变量赋初值通常是很方便的。在C++语言的声明中，这种初始赋值是自动的。也就是说，语句

```
LaserClass  Laser1, Laser2, Laser3;
```

不仅创建了变量Laser1、Laser2和Laser3，而且还创建了3个LaserClass“类型”的对象，其中每一个作为每个变量的值。在Java和C#中，这种初始赋值与对一个基本类型变量赋初值的方法基本相同。特别是，鉴于语句

```
int x = 3;
```

不仅声明了一个整型变量x，同时还为这个新的变量赋值为3，语句

```
LaserClass Laser1 = new LaserClass();
```

不仅声明了一个LaserClass“类型”的变量Laser1，而且还通过LaserClass类模板创建了一个新的对象，并且将其作为初值赋给Laser1。

在进一步讨论之前，我们应该强调一下类和对象之间的区别。类是一个模板，对象是由这个模板创建出来的。一个类能够用来创建许多个对象，我们经常将对象称为类（构建该对象的类）的**实例**（instance）。因此，在我们的计算机游戏中，Laser1、Laser2变量的值和Laser3都是LaserClass类的实例。

在用声明语句创建对象并且将其赋给变量 Laser1、Laser2、Laser3 之后，可以通过编写命令语句来激活这些对象（按面向对象术语来说，这称为“向对象发送消息”）中合适的方法。具体而言，我们可用赋给变量 Laser1 的对象通过以下语句执行 fire 方法：

```
Laser1.fire();
```

或者，我们可让赋给Laser2的对象执行它的turnLeft方法，这是通过语句

```
Laser2.turnLeft();
```

来实现的。这些实际上是过程的调用。的确，前面一个语句是调用赋给变量Laser1的对象内部的过程（方法）fire，后者则是调用赋给变量Laser2的对象内部的过程turnLeft。

在这个阶段，流星游戏例子已经给出了掌握典型面向对象程序总体结构的背景知识（图6-20）。它包含了与图6-19相似的一系列类的描述，每一个都描述了程序中使用的一个或多个对象的结构。另外，程序会包含一个命令程序段（通常和名字“main”有关），这个命令程序段包括了当程序运行时最初要执行的步骤序列。这个段包括与我们激光类的声明类似的声明语句，用来建立程序中使用的对象，还包括调用那些对象中执行方法的命令语句。

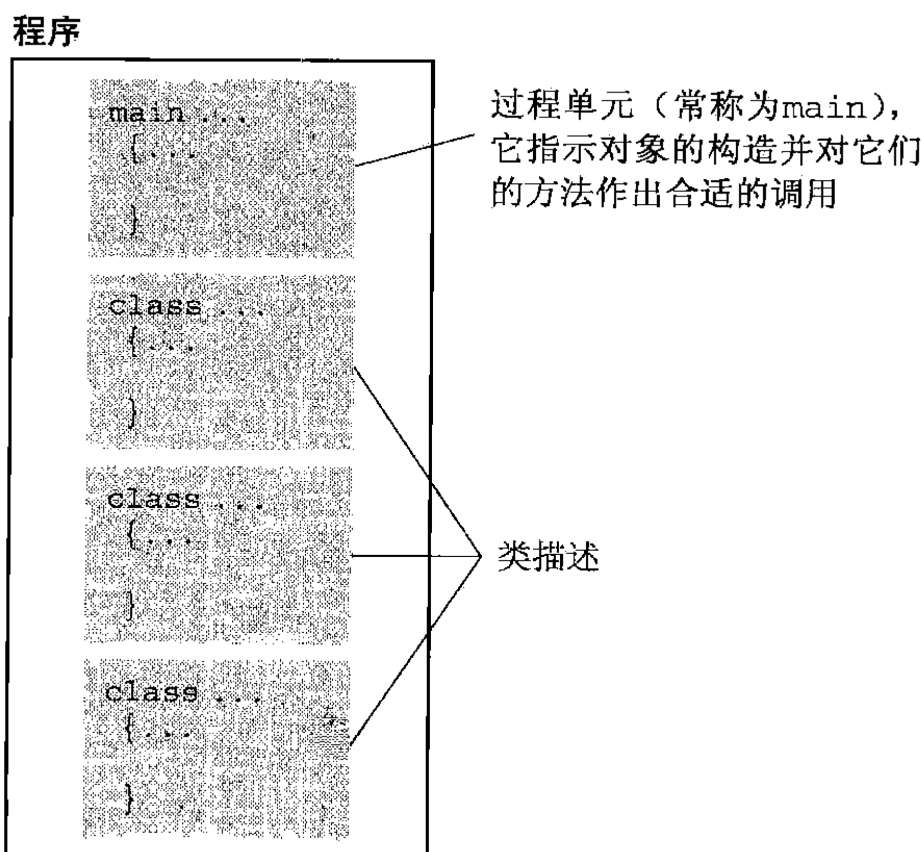


图6-20 典型的面向对象程序结构

6.5.2 构造器

当构造对象时，通常需要进行一些个性化的定制。例如，在我们的电脑游戏中，有可能需要实现一些具有不同初始能量设置的激光器，这就意味着不同对象中的 RemainingPower 实例变量应该给定不同的初始值。这种初始化通过在合适的类中定义特殊的方法——**构造器**（constructor）——来进行，它是在构建类的对象时自动执行的。一个构造器在类的定义中是通过使用与类名相同的名字来标识的。

图6-21给出了图6-19中的LaserClass类的扩展定义。注意，它包括一个构造器，该构造器的形式是名为LaserClass的方法。这个方法将它接受的参数值赋给实例变量Remaining Power。因此，当一个对象在类中构建出来时，这个方法就会执行，使得RemainingPower被初始化为一个合适的值。

```
class  LaserClass
{ int RemainingPower;

{ LaserClass (InitialPower)
  { RemainingPower = InitialPower;
  }

  void turnRight (  )
  { ... }

  void turnLeft (  )
  { ... }

  void fire (  )
  { ... }

}
```

当一个对象被创建时，构造器给Remaining-Power赋一个值

图6-21 带有构造器的类

构造器所使用的实参是在引起构建该对象的语句里的参数列表中标识的。因此，基于图 6-21 中给出的类的定义，C++程序员将会编写语句

```
LaserClass Laser1(50), Laser2(100);
```

来创建两个LaserClass类型的对象，一个是Laser1，初始能量值50，另一个是Laser2，初始能量值100。使用Java和C#的程序员完成同样工作的语句是：

```
LaserClass Laser1 = new LaserClass(50);
LaserClass Laser2 = new LaserClass(100);
```

6.5.3 附加特性

假设我们需要改进游戏，以使得玩家达到一定的分数时，可以获得奖励为其现有的激光器补充能量。除了可以补充能量以外，这些激光器与其他激光器有同样的属性。

为了简化对类似但不完全相同的对象的描述，面向对象语言允许一个类通过称为**继承**（inheritance）的方法包含其他类的属性。例如，假设使用 Java 来开发游戏程序，我们首先使用之前描述的类语句来定义类 LaserClass，这个类描述了程序中的所有激光器的共同属性。接下来，我们使用语句

```
class RechargeableLaser extends LaserClass
```

```
{
    .
    .
    .
}
```

来描述另一个类RechargeableLaser。(C++和C#语言用冒号来代替extends。)这里extends指明了这个类不仅继承了类LaserClass的特性，同时还包含了括号中出现的特性。括号可以包含新的方法（名字也许是recharge)，用它描述重置实例变量RemainingPower为其初始值的过程。一旦类定义好了，就可以使用语句

```
LaserClass Laser1, Laser2;
```

来将变量Laser1和Laser2声明为原来的激光器的变量，并且使用语句

```
RechargeableLaser Laser3, Laser4;
```

来将变量Laser3和Laser4声明为拥有类RechargeableLaser中描述的额外特性的激光器变量。

继承的使用导致各种相似但是不同的对象的存在，也因此导致了在 6.2 节中讨论的重载的现象。(让我们回忆一下，重载是使用一个诸如+的符号，根据操作数类型的不同代表了不同的操作。)假设一个面向对象的图形开发包包含各种对象，每一个都代表了一种形状（圆形、矩形、三角形等)。一个特定的图像可能由一组这类对象构成。每个对象都有自己的大小、位置和颜色，都有自己响应消息的方式，例如移动到一个新的位置或者在屏幕上画出自己。我们仅仅对图像中的每个对象发送“画自己”的消息来画图。但是，对象形状的不同决定了所使用的画一个对象的例程是不同的——一个正方形的画法和圆形的画法是不同的。这种对消息的自定义解释称为**多态**（polymorphism)，这种消息是多态的。

封装（encapsulation）是与面向对象程序设计相关的另一个特性，它是指限制对一个对象内部属性的访问。说一个对象的特定属性是封装的，就意味着只有对象自己才可以访问它们。被封装的属性称为私有属性，而可以从对象外部访问到的属性称为公有属性。

例如，让我们回到图 6-19 中的 LaserClass 类。它描述了一个实例变量 RemainingPower 以及 3 个方法 turnRight、turnLeft 和 fire。这些方法可以被其他程序单元访问，以使 LaserClass 的实例执行合适的动作。但是有关对 RemainingPower 值的修改应当只能由实例内部方法来实现，其他程序单元不应具有直接访问这个值的能力。为了强制实现这些规则，如图 6-22 所示，我们仅需要指定 RemainingPower 为一个私有变量而使其他 3 个方法都是公有的。通过这种设计，在程序编译期间任何试图从对象的外部对 RemainingPower 的值进行访问的操作都会被标识为错误，并要求程序员改正该错误。

问题与练习

1. 对象和类之间的区别是什么？
2. 在本节的计算机游戏中，除了LaserClass，还可以找到什么对象类？除了RemainingPower，还可在LaserClass类中找到什么实例变量？
3. 假设类PartTimeEmployee和FullTimeEmployee继承了类Employee的属性。你可以想到这两个类中都有什么特性？
4. 什么是构造器？
5. 为什么类中的一些项要设计为私有的？

类中的成分定义为公有或是私有依赖于是否要从其他程序单元访问它们

```
class  LaserClass
{private int RemainingPower;
 public LaserClass (InitialPower)
 {RemainingPower = InitialPower;
 }
 public void turnRight (  )
 { ... }
 public void turnLeft (  )
 { ... }
 public void fire (  )
 { ... }
}
```

图6-22　在Java和C#中使用封装的LasserClass的定义

*6.6　程序设计中的并发活动

假设我们要为多路攻击敌人飞船的计算机游戏设计一个生成动画的程序。一个处理方法就是只设计一个程序，用该程序来控制整个动画屏幕。这种程序将负责绘制每一个飞船（假设动画做得很逼真），这意味着该程序将必须掌握许多飞船的各自特征。另一种方法就是设计一个控制程序来控制单个飞船的动画，每个飞船的特征由参数决定，在程序执行的开始阶段给参数赋值。然后，动画可以通过创建这个程序的多个激活（activation）来构建，每一次激活都使用各自的一组参数。同时执行这些激活，我们就会得到有许多飞船从屏幕上同时飞过的假象。

这种多个激活的同时执行称为**并行处理**（parallel processing）或**并发处理**（concurrent processing）。真正的并行处理需要多个CPU，每个CPU都执行一个激活。当仅有一个CPU可用时，并行处理给我们的错觉是允许多个激活分享一个CPU的时间，其方式与通过多道程序设计操作系统来执行相似（参见第3章）。

许多现代计算机应用程序在并行处理环境里解决要比在传统的单指令序列环境里更容易实现。于是，较新的程序设计语言提供了表达并行计算所涉及的语义结构的语法。这种语言的设计需要对这些语义结构的识别以及描述它们的语法的开发。

每一种程序设计语言都试图从自己的角度来处理并行处理范型，结果产生了不同的术语。例如，“激活”这个非正式的称谓在Ada语言中称为**任务**（task），而在Java中称为**线程**（thread）。这就是说，在Ada程序中同时发生的动作是通过创建多个任务来执行的，而在Java程序中是通过创建多个线程来执行的。在这两种情况下，结果都是多个活动被生成和执行，其方式与多任务操作系统控制下的进程是一样的。我们将采用Java中的术语，把这种“进程”都称为线程。

也许，在涉及并行处理的程序中必须表达的大多数最基本动作就是创建新的线程。如果希望可以同时执行飞船程序的多个激活，那么就需要说明这一点的语法。这种产生新线程的处理方法通常与请求一个传统过程的执行类似。不同之处在于，在传统的环境中，请求过程激活的

程序单元在所请求的过程终止之前不再往下执行（回想图 6-8），而在并行环境中，请求程序单元在被请求过程执行任务的同时继续向下执行（图 6-23）。因此，要创建多个飞船飞过屏幕的场景，我们可以写一个主程序，该主程序只生成飞船程序的多个激活，每个激活都配有描述不同飞船特征的参数。

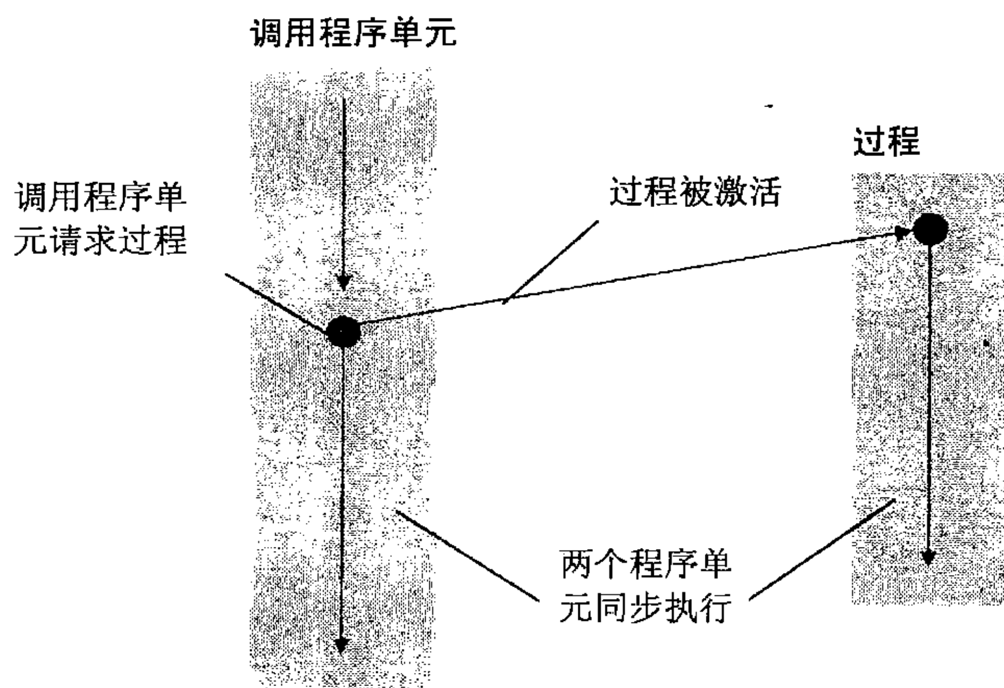


图6-23 产生线程

一个与并行处理相关的更复杂的问题就是处理线程之间的通信。例如，在飞船例子中，代表不同飞船的线程可能需要相互之间通告它们的方位以协调行动。在其他情况中，一个线程需要等待，直到另一个线程到达了它计算中的某个位置，或者一个线程在实现特定的任务之前需要停止另一个线程。

长期以来，这种通信需求一直是计算机科学家研究的课题，并且许多新的程序设计语言都有不同的解决线程之间交互问题的方法。例如，考虑当两个线程操作同一个数据时面临的通信问题。（这个例子在选读的3.4节更详细地进行了描述。）如果同时执行的两个线程都需要给一个公用的数据项加3，需要一个方法来保证在允许一个线程执行任务之前先允许另一个线程完成任务，否则它们会使用相同初始值开始各自的计算，这将意味着最后的结果将会是加3而不是加6。一次只能由一个线程访问的数据称为必须互斥访问的资源。

一种实现互斥存取的方法就是编写描述所涉及线程的程序单元，以便在一个线程正在使用共享数据时，它可以阻止其他线程访问这个数据，直到这样的访问是安全的。（这个方法在 3.4 节中已经描述过，在那里我们把一个进程中访问共享数据的那部分标识为临界区。）经验表明，这个方法是有缺陷的，它把保证互斥的任务分散在程序各处——每个访问该数据的程序单元都必须正确设计以确保这种互斥，因此一个程序段中的错误就能使整个系统崩溃。因此，许多人认为一个更好的解决办法是使数据有控制对自身访问的能力。简而言之，不再依赖于访问数据的线程来防止多重访问，而是赋予数据本身这个能力，结果是访问控制集中于程序中的一个点，而不是分散于许多程序单元中。增加了对自身访问的控制能力的数据项称作**监控程序**（monitor）。

我们看到，程序设计语言中对于并行处理的设计包括开发表达诸如线程的创建、线程的暂停和重启、临界区的标识以及监控程序的组成等方法。

在结束本节时，我们应当注意，尽管动画提供了一个探索并行计算问题的有趣场景，但是这仅仅是从并行处理技术中受益的许多领域中的一个。天气预报、空中交通管制、复杂系统（包括核反应、行人的交通等）的模拟、计算机网络以及数据库的维护都是可以应用这项技术的领域。

面向智能手机编程

面向手持设备、移动设备和嵌入式设备开发软件，通常使用在其他环境中使用的相同的通用编程语言。若有较大的键盘和额外的耐心，我们可以单使用智能手机编写某些智能手机应用。不过，大多数情况下，智能手机的软件是使用特殊软件系统在台式机上开发的，而这些特殊软件系统则提供用于编辑、翻译和测试智能手机软件的工具。简单的应用通常用Java、C++和C#编写。不过，若要编写较复杂的应用或核心系统软件，我们可能需要额外支持并行处理和事件驱动的编程。

问题与练习

1. 可以进行并发处理的程序设计语言有哪些特性是传统语言中没有的?
2. 描述两种可以确保对数据互斥访问的方法?
3. 说出除了动画以外可受益于并行计算的其他环境。

*6.7　说明性程序设计

在6.1节，我们断言形式逻辑提供了一个通用的解决问题的算法，围绕这个算法可以构建一个说明性程序设计系统。在本节中，我们将研究这个断言，首先介绍这个算法的基本原理，然后再简要地看一看基于这种算法的说明性程序设计语言。

6.7.1　逻辑推演

假设知道Kermit要么病了要么就在舞台上，并且被告知Kermit不在舞台上，我们就可以推断出Kermit一定是病了。这个演绎推论的例子称为**消解**（resolution）。消解是一种称为**推理法则**（inference rule）的许多技法之一，可以用来从大量的陈述中推导出结果。

为了更好地理解消解，我们首先可以用单个字母表示简单命题，通过符号¬来表示命题的否定。例如，我们用A来代表“Kermit是一位王子”，用B来代表“Piggy小姐是一位演员”，那么，表达式

A OR B

意味着“Kermit是一位王子或者Piggy小姐是一位演员”，而

B AND $\neg A$

意味着“Piggy小姐是一位演员而Kermit不是一位王子”。我们将用箭头来表示蕴涵关系。例如，表达式

$A \rightarrow B$

意味着“如果Kermit是一位王子，Piggy小姐就是一位演员”。

以这种通式，消解原理意味着从命题

P OR Q

和

R OR $\neg Q$

可以归结出命题

P OR R

这样，我们就说原来的两个命题消解形成了第三个命题，我们称之为**消解式**（resolvent）。重要的是要看到，这个消解式是原始命题的逻辑结论。这就是说，如果原始命题是真的，那么消解式也一定是真的。（如果Q是真的，那么R一定是真的；如果Q是假的，那么P一定是真的。因此，不管Q是真是假，P或者R一定是真的。）

如图 6-24 所示，我们用图形化的方法表示了这两个命题的消解，在这个图中，引自原始命题的连线指向下面的消解式。注意，消解只能用于成对命题，并且这些命题以**子句形式**（clause form）出现——也就是说，命题的基本组件是通过布尔运算符 OR 连接起来的。因此

P OR Q

是子句形式，而

$P \rightarrow Q$

就不是子句形式。这对于我们来说不是很重要，因为这是数理逻辑中的一个定理的推导结果，这个定理说，任何以一阶谓词逻辑（一个用扩充的表达能力表示语句的系统）表达的语句都可以用子句形式来表达。我们不在这里进一步探讨这个重要的定理，但是为了今后的使用，我们发现命题

$P \rightarrow Q$

等价于子句形式的命题

Q OR $\neg P$

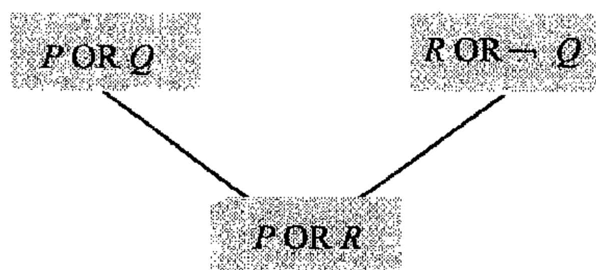


图6-24 消解命题（P OR Q）和（R OR $\neg Q$）推出（P OR R）

如果一组命题中的所有命题不可能同时为真，那么这组命题就是**不相容的**（inconsistent）。换句话说，一组不相容的命题是一组自相矛盾的命题。一个简单的例子是命题P与命题$\neg P$的组合。逻辑学家已经证明，重复的消解提供了验证一个不相容子句的集合的不相容性的系统化方法。这个方法就是，如果反复进行消解而产生了一个空子句（消解命题P和命题$\neg P$的结果），那么原来的一组命题必定是不相容的。例如，图6-25证明命题集合

P OR Q　　R OR $\neg Q$　　$\neg R$　　$\neg P$

是不相容的。

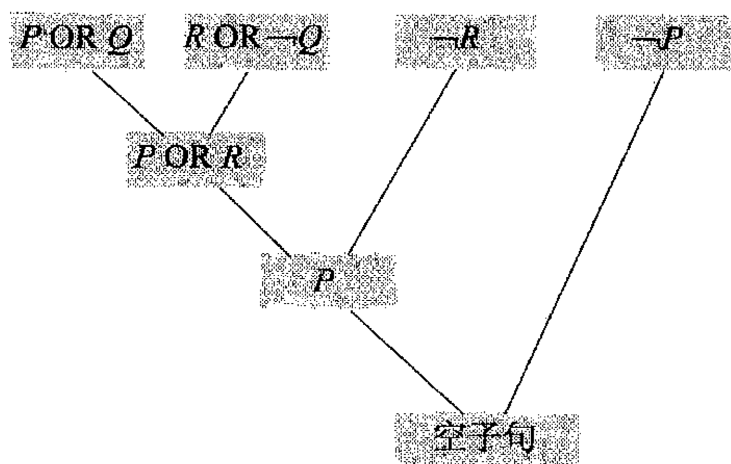


图6-25 消解命题（P OR Q）、（R OR $\neg Q$）、$\neg R$和$\neg P$

假定我们要证明一组命题蕴涵了命题P。推导这个命题P就相当于对命题$\neg P$取反。因此，我们所需要做的就是将原来的命题与$\neg P$进行消解，直到产生一个空子句。基于获得的空子句，我们就可以说$\neg P$与原来的命题组是不相容的，从而可以推导出原来的一组命题一定蕴涵P。

在将消解应用于一个实际的程序设计环境之前，还有最后一个问题。假设我们有两个命题

```
(Mary is at X)→ (Mary's lamb is at X)
```

（其中X代表任何地方）和

```
Mary is at home
```

按照子句形式，这两个命题变为

```
(Mary's lamb is at X) OR ¬(Mary is at X)
```

和

```
(Mary is at home)
```

乍一看，似乎没有可以消解的元素。另一方面，元素（Mary is at home）和¬（Mary is at X）近于相互对立。问题是要认识到，命题 `Mary is at X` 是一个关于位置的通用命题，而关于 home 的命题则是它的特殊形式。因此，对于第一个命题的特殊情况是：

```
(Mary's lamb is at home)OR ¬(Mary is at home)
```

它可以与命题

```
(Mary is at home)
```

消解产生命题

```
(Mary's lamb is at home)
```

把值赋给变量（例如将home赋给X），从而使得消解可以进行的过程称为**单一化**（unification）。这个过程使得演绎系统中一般的命题可以用于特定的应用。

6.7.2 Prolog

程序设计语言 Prolog（PROgramming in LOGic 的缩写）是一个说明性程序设计语言，它解决问题的基本算法就是反复地进行消解。这样的语言称为**逻辑程序设计**（logic programming）语言。一个 Prolog 程序由一组初始语句组成，基本的算法在它们之上进行演绎推理。构成这些语句的成分称为**谓词**（predicate）。一个谓词由一个标识符和一个带括号的语句组成，括号里列有该谓词的变元。谓词代表了与它的变元相关的事实，因而谓词标识符的选取通常都反映该事实的基本语义。因此，如果希望表达 Bill 是 Mary 的家长，我们可以使用这样的谓词形式：

```
parent(bill, mary)
```

注意，尽管这个谓词里的变元表示正常的名字，但是它们是以小写字母开始，这是因为Prolog区分常量和变量的依据是常量以小写字母开始，而变量以大写字母开始。[这里，我们使用了Prolog的专有名词，用术语常量（constant）代替更通用的术语*字面量*（literal）。更准确一点讲，在Prolog里用名词`bill`（注意是小写）表示的字面量可能会被更通用的表示法表示为`Bill`。名词`Bill`（注意B是大写）在Prolog中表示变量。]

一个 Prolog 程序中的语句有事实和规则两种，每个都是以一个句点来结束。一个事实包括一个谓词。例如，乌龟比蜗牛快这个事实用 Prolog 语句可以这样来描述：

```
faster(turtle, snail).
```

而兔子比乌龟快的事实可以这样来表示：

```
faster(rabbit, turtle).
```

一个 Prolog 规则是一个“蕴涵”语句。但是，Prolog 程序员并不是将语句写成像 $X \rightarrow Y$ 这样，而是写成“Y if X”这样，除非使用符号:-（一个冒号和一个连字符）来替代符号 if。因此规则“X is old implies X is wise”对于一个逻辑学家来说可能要这样来表述：

```
old(X)→wise(X)
```

而在Prolog语言里表示为：

```
wise(X):- old(X).
```

又如，规则

```
(faster(X, Y) AND faster(Y, Z) ) → faster(X, Z)
```

在Prolog里表达为：

```
faster(X, Z):- faster(X, Y),faster(Y, Z).
```

这个分隔 faster (X,Y) 和 faster (Y,Z) 的逗号代表合取符 AND。尽管这样的规则不是子句形式，但是它们在 Prolog 里是被允许的，因为它们很容易转化为子句形式。

记住，Prolog 系统并不知道程序中谓词的意义，它只是根据消解法则以完全符号的方式来对语句进行操作。因此，用事实和规则来描述谓词的有关特性完全是程序员的职责。从这一点来看，Prolog 的事实倾向于用来标识谓词的特殊实例，而规则用来描述一般的法则。这就是前面有关谓词 faster 的语句所使用的方法。这两个事实描述了“快”的特定实例，而规则描述了一个一般的属性。注意，兔子比蜗牛快的事实，尽管没有明说，但这是两个事实通过规则结合的结论。

当使用 Prolog 语言进行软件开发时，程序员的工作就是开发一组事实和规则来描述已知的信息。这些事实和规则构成了要在演绎推理系统中使用的初始语句集合。一旦这个语句集合确定了，那么可以向系统建议一些猜测（在 Prolog 术语中称为目标）——通常通过键盘输入它们。当这样的一个目标向 Prolog 系统提交后，系统利用消解来试图证明这个目标是初始语句的推导结果。基于描述 faster 关系的一组语句，下面的每一个目标

```
faster(turtle, snail).
faster(rabbit, turtle).
faster(rabbit, snail).
```

都可以证明，因为每一个都是初始语句的逻辑结果。最开始的两个与出现在初始语句中的事实相同，而第三个需要系统的某种程度上的演绎。

如果我们提供的目标的变元不是常量而是变量，那么可以得到更为有趣的例子。在这种情况下，Prolog试图从初始语句中推导出目标，同时跟踪推导所需要的单一化。然后，如果这个目标达到了，那么Prolog将报告这些单一化。例如，考虑目标：

```
faster(W, snail).
```

Prolog对于它的响应是报告：

```
faster(turtle, snail).
```

的确，这是初始语句的一个结论，并且通过单一化与这个目标一致。此外，如果要求Prolog提供更多的结论，那么它会找到并报告下面的结论：

```
faster(rabbit, snail).
```

而我们能通过提出目标

```
faster(rabbit, W).
```

要求Prolog寻找一些比兔子慢的动物的实例。事实上，如果我们以目标

```
faster(V, W).
```

开始，Prolog将会报告所有可以从初始语句中推导出来的faster关系。这意味着一个简单的Prolog程序可以用来证明某种特定的动物比另一种快，找出那些比某种给定的动物快的动物，找出那些比某种给定的动物慢的动物，或者找出所有faster关系。

这个潜在的多功能性是激发了计算机科学家想象的特性之一。遗憾的是，当在Prolog系统中实现时，消解过程显示了它理论形式中并没有呈现出来的限制，因此Prolog程序可能不能满足它被预期的灵活性要求。为便于理解，首先注意图6-25中的示意图只显示了与手头任务相关的那些消解，当然还有其他一些消解方式。例如，最左和最右子句的消解可产生消解式Q。这样，除了描述应用所涉及的事实和规则的语句外，Prolog程序经常必须包含额外的语句，这些语句的作用是正确地指导消解过程。由于这个原因，实际的Prolog程序可能不能获得我们先前例子所暗示的目的多样性。

问题与练习

1. 语句R、S、T、U和V中哪一个是（$\neg R$ OR T OR S）、（$\neg S$ OR V）、（$\neg V$ OR R）、（U OR $\neg S$）、（T OR U）和（S OR V）构成的集合的逻辑结果？
2. 下面的语句集合是相容的吗?为什么？
   P OR Q OR R　　$\neg R$ OR Q　　R OR $\neg P$　　$\neg Q$
3. 完成下面的Prolog程序末尾的两个规则，以使得谓词mother(X, Y)的含义是“X是Y的母亲”，而father(X,Y)表示“X是Y的父亲”。
```
female(carol).
female(sue).
male(bill).
male(john).
parent(john, carol).
parent(sue, carol).
mother(X, Y):-
father(X, Y):-
```
4. 根据问题与练习3中的Prolog程序，下面的规则想要表述这样的含义，即如果X和Y有共同的父母，X是Y的同胞。
```
sibling(X, Y) :- Parent(Z, X), Parent(Z, Y).
```
   这一同胞关系定义会使Prolog得出什么意外的结论？

复习题

（带*的题目涉及选读小节的内容。）

1. 一种程序设计语言是机器独立的，这意味着什么？
2. 将下面的伪代码程序翻译成附录C中描述的机器语言：
```
x ← 0;
while(x < 3)do
   (x ← x + 1)
```
3. 将语句
```
Halfway←Length + Width
```
   翻译成附录C中描述的机器语言，假设Length、Width和Halfway都以浮点型表示。
4. 将高级语言语句
```
if(X equals 0)
   then Z ← Y + W
   else Z ← Y + X
```

翻译成附录C中描述的机器语言，假设W、X、Y和Z的值都用二进制补码表示，每个使用存储器中的一个字节。

5. 在第4题中，为了翻译这些语句，为什么标识变量的数据类型是必要的？为什么许多高级程序设计语言需要程序员在程序的开始位置标识每一个变量的类型？
6. 说出并描述4种不同的程序设计范型。
7. 假设函数f需要两个数值作为它的参数，并且它要返回这两个数中较小的一个作为输出值。如果w、x、y和z都代表数值，$f(f(w, x), f(y, z))$的返回结果是什么？
8. 假设函数f返回的结果是输入的字符串的反序，并且函数g返回输入的两个字符串的连接。如果x是字符串abcd，那么$g(f(x), x)$的返回值是什么？
9. 假设你要写一个面向对象程序来维护自己的财务记录。在表示活期账户的对象里应该存放什么数据？这个对象会响应什么样的消息？程序中可能使用的其他对象是什么？
10. 概述机器语言和汇编语言的区别。
11. 为附录C中描述的机器语言设计一个汇编语言。
12. 程序员John认为在程序中声明一个常量的功能是不必要的，因为可以用一个变量来代替它。例如，在6.2节中的AirportAlt的例子中，可以把AirportAlt声明为一个变量，然后在程序开始的时候为其赋需要的值。为什么这种方法不如使用常量好？
13. 概述声明性语句和命令型语句之间的区别。
14. 解释字面量、常量和变量之间的区别。
15. a. 什么是运算符优先级？
    b. 根据运算符优先级，表达式6+2×3的值是什么？
16. 什么是结构化程序设计？
17. 语句

    ```
    if (X = 5) then (...)
    ```
    中的等号和赋值语句
    ```
    X = 2 + Y
    ```
    中的等号的区别是什么？
18. 画一个流程图来表示下面的for语句所表达的结构。

    ```
    for(int x = 2; x < 8; ++x)
    {...}
    ```
19. 用第5章伪代码中的while语句把下列的for语句翻译成等效的程序段。

    ```
    for(int x = 2; x < 8; ++x)
    {...}
    ```
20. 如果你熟悉乐谱，像对待程序设计语言那样分析音乐符号。什么是控制结构？什么是插入程序注释的语法？什么音乐符号有类似于图6-7中的for语句的语义？
21. 画一个流程图来表示下面的语句所表达的结构。

    ```
    switch (suit)
      {case "clubs": bid(1);
       case "diamonds": bid(2);
       case "hearts": bid(3);
       case "spades": bid(4);
     }
    ```
22. 重写下面的程序段，使用一个case语句代替嵌套的if-then-else语句。

    ```
    if (W = 5)
       then(Z ← 7)
       else (if (W = 6)
               then(Y ← 7)
               else (if (W = 7)
                        then(X ← 7)
                    )
            )
    ```
23. 使用一个if-then-else语句，概括下面的例程：

    ```
       if X > 5 then goto 80
       X = X + 1
       goto 90
    80 X = X + 2
    90 stop
    ```
24. 为了实现下述每个活动，概述命令型语言和面向对象语言中的基本控制结构。
    a. 判断将要执行哪一条命令。
    b. 重复一组命令。
    c. 改变变量的值。
25. 概述翻译器和解释器的区别。
26. 假设程序中的变量x声明为整型。当执行语句
    ```
    X ←2.5
    ```
    时，会发生什么错误？
27. 说一个程序设计语言是强类型的意味着什么？
28. 为什么一个大的数组不太可能通过按值传递的方式传递给过程？
29. 假设过程Modify用第5章的伪代码定义为

```
procedure Modify(Y)
  Y←7;
  打印Y的值
```

如果参数是按值传递的，当执行下面这个程序段时，会打印出什么结果？如果参数是按引用传递呢？

```
X←5;
对X应用Modify过程;
打印X的值;
```

30. 假设过程Modify用第5章的伪代码定义为

```
procedure Modify(Y)
 Y←9;
  打印X的值;
  打印Y的值;
```

假设X是全局变量。如果参数按值传递，那么执行下面的程序段会打印出什么结果？如果参数是按引用传递呢？

```
X←5;
对X应用Modify过程;
打印X的值;
```

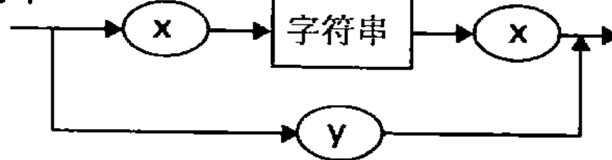
31. 有时，把实参传递给一个过程时是通过产生一个副本给过程使用（如按值传递时）实现，但在过程完成时，过程的副本里的值在调用过程继续执行之前传递给实参。在这种情况下，称参数是按值-结果传递的。如果参数按值-结果传递，那么第30题的程序段会打印出什么结果？
32. a. 按值传递相对于按引用传递有哪些优点？
    b. 按引用传递相对于按值传递有哪些优点？
33. 语句X←3+2×5存在什么歧义？
34. 假设一个小公司有5名雇员，并且计划增加雇员数目到6名。假设下面的赋值语句是该公司一个程序中的语句：

```
DailySalary= TotalSal/5;
AvgSalary = TotalSal/5;
DailySales = TotalSales/5;
AvgSales = TotalSales/5;
```

那么，如果原程序使用了NumberOfEmp和WorkWeek两个常量（值都为5），如何简化更新程序的任务才能使赋值语句可表达为：

```
DailySalary = TotalSal/DaysWk;
AvgSalary = TotalSal/NumEmpl;
DailySales = TotalSales/DaysWk;
AvgSales = TotalSales/NumEmpl;
```

35. a. 形式语言和自然语言之间的区别是什么？
    b. 分别举例。
36. 用语法图来表示第5章的伪代码中的while语句的结构。
37. 设计一组语法图来描述你所在区域的电话号码的语法。例如在美国，电话号码由分区电话号码、地区电话号码和一个4位数字组成，如（444）555-1234。
38. 设计一组语法图来描述你的母语里的一个简单的句子。
39. 设计一组语法图来描述不同的表示日期的方法，如“月/日/年”或是“月/日，年”
40. 设计一组语法图来描述下述句子的文法结构：在yes的后面有与yes个数相同的no。例如句子“yes yes no no”符合要求，而句子“no yes”、“yes no no”和“yes no yes”就不满足要求。
41. 有一种句子是这样的：在yes的后面有与yes个数相同的no，在其后又有相同数目的maybe，例如，“yes no maybe”、“yes yes no no maybe maybe”就是这样的句子，而“yes maybe”、“yes no no maybe maybe”、“maybe no”都不是。给出一个论据说明这种句子的文法结构的语法图的集合不能被设计出来。
42. 写一个句子描述下面语法图所定义的字符串的结构，然后画出字符串xxyxx的语法分析树。



43. 为6.4节中的问题5增加语法图，以得到一个定义Dance结构为Chacha或者为Waltz的一组语法图，其中Waltz包括一个或多个以下模式的副本：

forward diagonal close

或

backward diagonal close

44. 基于图6-15中的语法图，为表达式x×y+y÷x画出语法分析树。
45. 当为下面的语句生成机器代码时，代码生成器可以实现哪些代码优化？

```
if(X = 5) then (Z ← X + 2)
          else (Z ← X + 4)
```

46. 简化下面的程序段：

```
Y ← 5;
if(Y = 7)
  then(Z ← 8)
  else(Z ← 9)
```

47. 简化下面的程序段；

```
while (X not equal to 5)do
    (X←5)
```

48. 在面向对象程序设计环境中，类型和类有哪些相似，又有哪些不同？
49. 描述不同类型建筑的类的开发是如何使用继承的？
50. 在类中私有部分与公有部分的区别是什么？
51. a. 给出一个实例变量应该是私有的例子。
    b. 给出一个实例变量应该是公有的例子。
    c. 给出一个方法应该是私有的例子。
    d. 给出一个方法应该是公有的例子。
52. 说明在模拟酒店门厅里行人交通时可能需要的一些对象以及某些对象需要实现的动作。

*53. 在程序设计语言环境中，术语“监控程序”指什么？

*54. 并发处理的什么属性使它需要使用支持并发的程序设计语言？

*55. 画一个表示消解的图（类似于图6-25），来说明语句（Q OR $\neg R$）、（T OR R）、$\neg P$、（P OR $\neg T$）和（P OR $\neg Q$）的集合是不相容的。

*56. 语句$\neg R$、（T OR R）、（P OR $\neg Q$）、（P OR $\neg T$）和（R OR $\neg P$）的集合是相容的吗？解释你的答案。

*57. 扩展6.7节中问题3和4描述的Prolog程序，以包含另外的家庭关系，如叔叔、阿姨、祖父母和堂兄弟。还要增加定义`parent(X, Y, Z)`的规则，`parent(X, Y, Z)`的意思是：X和Y是Z的父母。

*58. 假设下列Prolog程序的第一条语句意味着“Alice喜欢运动”，翻译程序中的最后两个语句。然后，基于这个程序，列出Prolog将能得出的Alice喜欢的所有事情。

```
likes(alice,sports).
likes(alice,music).
likes(carol,music).
likes(david,X)  :- likes(X,sports).
likes(alice,X)  :- likes(david,X).
```

*59. 如果下面的程序段在一台用1.7节中描述的8位浮点格式表示数值的机器上执行，会遇到什么问题？

```
X ← 0.01;
while(X not equal to 1.00) do
  (print the value of X;
   X ← X + 0.01)
```

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 通常，版权法支持与一个想法的表达有关的所有权，而不是这个想法本身。因此，一本书中的段落是受版权法保护的，但是这一段落表述的思想就不受保护。这种权利如何应用到源程序和它表达的算法呢？一个了解某商业软件中所使用的算法的人，应当在多大程度上被允许编写表达这些相同算法的程序，并将这个软件推向市场？
2. 通过使用高级程序设计语言，程序员可以使用诸如`if`、`then`和`while`这样的单词来表达算法。计算机理解这些词的含义到了什么程度？正确应对这些词的使用的能力是否意味着对词语的理解？你怎么知道另一个人理解了你所说的？
3. 一个开发新的有用的程序设计语言的人应当有从这个语言的使用中获利的权利吗？如果有，如何保护这样的权利？一种程序设计语言可以在多大程度上被拥有？公司对雇员创新的智力成果有多大程度上的所有权？
4. 在临近最后期限时，一个程序员打算放弃用注释语句编制文档以使程序能够按时完成，这可以被接受吗？（初学者在得知文档对于专业的软件开发人员是如何重要时，往往非常惊讶。）
5. 许多程序设计语言的研发致力于开发出这样的语言，它可以使程序员编写出人类易读且容易理解的程序。在多大程度上应当要求程序员使用这些能力？也就是说，对于能够正

确实现功能但从人的角度看来写得不好的程序，在什么程度上才算是好程序？

6. 假设一个业余程序员写了一个程序供自己使用。这个程序没有使用程序设计语言的易读特性，它也不够高效，并且包含了利用特殊情况（这个程序员试图使用这个程序的特殊环境）的省事方法。此后，这个程序员把他的程序复制给希望使用这个程序的朋友，而他的朋友又把这个程序复制给了他们的朋友。这个程序员要为他的程序可能出现的问题负多大责任？
7. 计算机专业人员对于各种程序设计范型应该精通到什么程度？某些公司坚持在公司的所有软件开发中都使用同一种预先确定好的程序设计语言。如果某计算机专业人员在这种公司工作，你对前面问题的回答是否会发生变化？

课外阅读

Aho, A. V., M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*, 2nd ed. Boston, MA: Addison-Wesley, 2007.

Barnes, J. *Programming in Ada 2005*, Boston, MA: Addison-Wesley, 2006.

Clocksin, W. F. and C. S. Mellish. *Programming in Prolog*, 5th ed. New York: Springer-Verlag, 2003.

Friedman, D. P., and M. Felleisen. *The Little Schemer,* 4th ed. Cambridge, MA: MIT Press, 1995.

Hamburger, H. and D. Richards. *Logic and Language Models for Computer Science*. Upper Saddle River, NJ: Prentice-Hall, 2002.

Kernighan, B.W., and D.M. Ritchie. *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

Metcalf, M. and J. Reid. *Fortran 90/95 Explained*, 2nd ed. Oxford, England: Oxford University Press, 1999.

Pratt, T. W. and M. V. Zelkowitz. *Programming Languages, Design and Implementation,* 4th ed. Upper Saddle River, NJ: Prentice-Hall, 2001.

Savitch, W. *Absolute C++*, 3rd ed. Boston, MA: Addison-Wesley, 2008.

Savitch, W. *Absolute Java*, 3rd ed. Boston, MA: Addison-Wesley, 2008.

Savitch, W. *Problem Solving with* C++, 6th ed. Boston, MA: Addison-Wesley, 2008.

Scott, M. L. *Programming Language Pragmatics,* 3rd ed. New York: Morgan Kaufmann, 2009.

Sebesta, R. W. *Concepts of Programming Languages*, 9th ed. Boston, MA: Addison-Wesley, 2009.

Wu, C. T. *An Introduction to Object-Oriented Programming with Java,* 3rd ed. Burr Ridge, IL: McGraw-Hill, 2008.

第7章

软 件 工 程

本章讨论的是在开发大型的复杂软件系统过程中遇到的问题。之所以将这门学科称为软件工程，是因为软件开发是一个工程化的过程。研究软件工程的目标就是要找到一种原则，能够指导软件开发过程，进而生产出高效的、可靠的软件产品。

本章内容

软件工程是计算机学科中的一个分支，致力于寻找指导大型复杂的软件系统的开发原则。开发这类系统所面对的问题并非只是编写小程序所面对问题的放大。比如说，开发大型系统的时候，要求许多人工作很长时间，而在这期间，预期的系统需求可能会改变，参与该项目的人员也可能会变动。因此，软件工程包括了诸如人员管理和项目管理之类的主题，这样的主题更多与业务管理相关，而不是与计算机科学相关。当然，我们的侧重点还是放在那些与计算机科学密切相关的主题上。

7.1 软件工程学科

为了有助于理解软件工程中涉及的问题，这里可以想象构造一个大型的复杂设施（一辆汽车、一幢多层办公大楼或者一座教堂），对此进行设计，然后监管其构建过程。如何估算完成该项目所需的时间、费用以及其他资源？如何把项目分割成几个便于管理的模块？如何保证构建的模块相互协调一致？如何使工作在不同模块的人员相互沟通？如何衡量进度？如何妥善处理更广泛的细节问题（如门把手的选择、壁饰的设计、彩色玻璃窗的蓝色玻璃的需求量、柱子的强度、供暖系统的管道设计等）？在一个大型软件系统的开发过程中，同样需要面对如此繁多的问题。

有人也许会这样认为，工程是一个很成熟的领域，因此一定会有大量现成的工程技术可以用来解决软件工程中的这些问题。这种推理有一定的道理，但是忽略了软件的特性与其他工程领域特性之间存在着的本质上的不同。这些差别已经影响了软件工程项目，导致了其花费增加、推迟交付软件产品和软件产品不能满足用户的需求等后果。所以，在发展软件工程学科上，首先要做的工作是弄清这些差别。

差别之一涉及通过常用的预先定制的构件来构建系统的能力。一些传统的工程领域已经长期受益于这种能力，即在构建复杂的设备时，采用各种现成构件。例如，设计一辆新车时，没有必要重新设计引擎和传感器，利用这些构件以前的设计方案即可。然而，软件工程在这一点上却是很落后的。过去，以前设计的软件构件一般用于特定的领域。也就是说，这些构件本质上是为专门的应用设计的，所以将它们作为通用构件来使用是受限的。因此，复杂的软件系统历来都是从头做起。正如在这一章中我们将看到的那样，在这一点上已经取得了重要的进展，尽管还有很多工作要做。

软件工程与其他工程学科间的另一个差别在于缺少度量技术——**度量学**（metrics）——来衡量软件的属性。例如，为了计算开发一个软件系统的费用，人们希望能够估算出预期产品的复杂度，但是软件的复杂度估算方法还不太成熟。同样，评价软件质量的方法现在也不太成熟。对于机器设备，质量的重要量度是平均无故障时间，这是对设备耐损耗性的一个基本衡量指标。相反，软件没有损耗，所以这种方法在软件工程中并不适用。

软件指标不能以定量的方式测量，这也是软件工程和机械、电子工程不同，至今还未找到一个严格、坚实的立足点的主要原因。这些早些的学科（如机械和电子工程）是建立在成熟的物理学科的基础上的，然而软件工程仍然在找寻其自身的根基。

因而，现在的软件工程研究在两个层面上进行：一部分研究者（有时也称为实践派）的工作指向开发直接应用的技术；另一部分研究者（称为理论派）则致力于探寻软件工程的基础原理和理论，为将来构建更坚实的技术而努力。基于自身的原因，实践派以前开发和提出的许多方法已经被其他方法代替，新的方法可能也将随着时间的推移而淘汰。与此同时，理论派的进展也是一直很缓慢。

对实践派和理论派的两方面的进展需求是巨大的。我们这个社会已经沉迷于计算机系统及其相关的软件。我们的经济、保健、政府、法律实施、交通运输以及国防系统等都依赖于大型的软件系统。然而，在这些系统中，可靠性依然是最主要的问题。软件错误已经导致了一些大的灾难，新近的灾难如月亮的升起被误以为是核攻击、纽约银行造成的一天损失500万美元、空间探测器的失踪、过量的辐射导致人员的伤残，还有电话通信在同一时间大面积的瘫痪等。

这并不是说情况都很悲观。我们已经在解决诸如缺少预制的构件和衡量标准等问题方面取得很多进展。此外，由于计算机技术在软件开发过程中的应用，导致了称为**CASE**（Computer-Aided Software Engineering，计算机辅助软件工程）的出现，这使软件开发流程化，从而简化了软件的开发过程。CASE已经促进了许多计算机化系统的发展，这些系统称为**CASE工具**（CASE tool），包括项目设计系统（用来辅助经费预算、项目调度以及人员分配等）、项目管理系统（用来辅助监控项目的开发进度）、文档工具（用来辅助编写和组织文档）、原型与仿真系统（用来辅助开发原型系统）、界面设计系统（用来辅助图形用户界面的开发）、编程系统（用来辅助编写和调试程序）等。其中一些工具的功能和字处理程序、电子制表软件、电子邮件通信系统等差不多，最开始是为一般的应用开发的，已为软件工程师所采用。另外的一些工具主要是为软件工程环境专门定制的复杂软件包。实际上，称为**IDE**（Integrated Development Environment，集成开发环境）的系统把软件开发工具（编辑器、编译器、调试工具等）组合到单个集成的程序包中。为智能手机开发应用的系统便是这类系统的典型代表。这类系统不仅提供编写和调试软件所必需的编程工具，而且提供模拟器（借助于图形显示）让程序设计人员查看正在开发的软件在手机上的实际执行情况。

除了研究人员，专业人士和标准化组织（包括ISO）的努力，美国计算机协会（ACM）和美国电气及电子工程师协会（IEEE）也已经加入到改善软件工程状态的挑战中。这些努力包括：采用职业行为规范和道德规范来增强软件开发人员的职业精神，反对对个人职责的漠视态度；

建立衡量软件开发组织质量的标准，提供帮助这些组织改善它们标准的指导方针。

美国计算机协会

美国计算机协会（ACM）成立于1947年，是致力于推动艺术、科学及信息技术应用的国际性科学与教育组织。其总部在纽约，下设许多专业的工作组（SIG），分别致力于计算机体系结构、人工智能、生物医学计算、计算机与社会、计算机科学教育、计算机图形学、超文本/超媒体、操作系统、程序设计语言、仿真与建模和软件工程等主题。ACM的网站地址是http://www.acm.org。其职业道德和行为准则可在http://www.acm.org/constitution/code.html中找到。

美国电气及电子工程师协会

美国电气及电子工程师协会（IEEE，读作"i-triple-e"）是一个电气、电子和制造工程师的组织，成立于1963年，由美国电气工程师协会（由包括爱迪生在内的电气工程师于1884年创建）和美国无线电工程师协会（创建于1912年）合并而成。今天，IEEE的执行中心位于新泽西州的皮斯卡塔韦。协会由许多技术分会组成，如航天和电子系统协会、激光和光电子协会、机器人和自动化协会、交通运输技术协会以及计算机协会。IEEE也参与了各种标准的开发与制定。例如，IEEE的努力产生了今天在大多数计算机上使用的用单精度浮点数和双精度浮点数格式表示值的标准。

IEEE的主页地址是http://www.ieee.org，IEEE的计算机协会的主页地址是http://www.computer.org，IEEE的道德规范标准的主页地址是http://www.ieee.org/about/ whatis/code.html。

本章的其余部分将讨论软件工程的一些基本原理（如软件的生命周期和模块化等），预测软件工程发展的一些动向（如设计模式的定义与应用以及可复用软件构件的出现等），以及考察面向对象范型对这个领域产生的影响。

问题与练习

1. 为什么一个程序中的代码行的数量并非是对程序复杂性的一种好的度量？
2. 提出一种测量软件质量的量度建议，并说明这种量度有什么缺点？
3. 什么样的技术能用来确定一个软件单元中有多少错误？
4. 列举出两个在软件工程领域已经或当前正在改善的应用环境。

7.2 软件生命周期

软件工程最基础的概念就是软件生命周期。

7.2.1 周期是个整体

图7-1表示的是软件的生命周期。这个图表明了一个事实，即软件一旦开发完成，它就进入了一个既被使用又被维护的循环，这个循环将永不停止，直至软件生命周期结束。这种模式在许多工业产品中很常见。不同之处在于，对于其他产品，维护阶段往往是一个修复过程，而对于软件，维护阶段往往包括改错和更新。实际上，软件进入维护阶段，是由于以下的原因：发现了错误，软件应用中发生的变化需要在软件中做相应的修改，或者上一次修改中的变更导致软件中其他地方出现了问题。

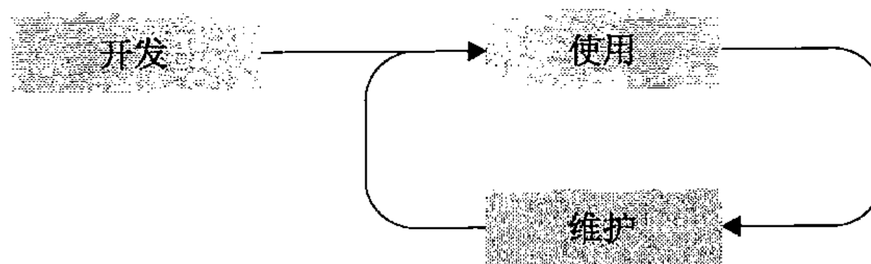


图7-1　软件的生命周期

无论软件因为什么样的原因进入维护阶段，这个过程都要求人员（通常不是原作者）研究底层的程序及其文档，直至把这个程序（或者至少是程序的相关部分）理解清楚。否则，任何的改动只会带来更多的问题。即使软件设计精良并有良好的文档，要达到这种理解也是一件困难的事情。事实上，到了这个阶段，软件的某部分往往会因为从头开发一个新系统要比成功修改现存的软件包要更容易这样一个借口而弃之不用（这个借口通常是真实的）。

经验表明，在软件开发期间稍作努力，就可能会在需要对软件进行修改时产生很不同的后果。例如，在第 6 章对数据描述语句的讨论中，我们可以看出使用常量与使用字面量相比会大大简化未来的修改工作。结果是，软件工程的大部分研究工作集中在软件生命周期的开发阶段，以利用这种付出与收益之间的杠杆作用。

7.2.2　传统的开发阶段

软件生命周期传统开发阶段的主要步骤是需求分析、设计、实现和测试（参见图 7-2）。

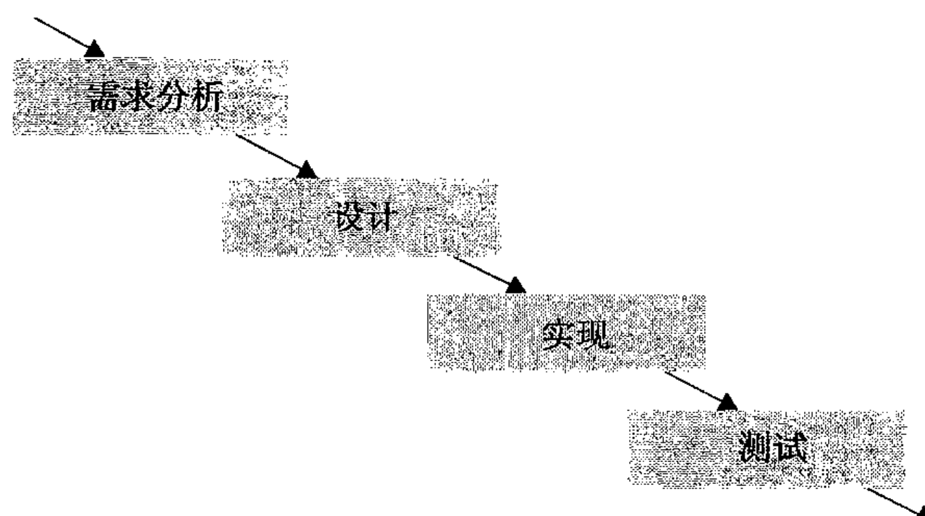


图7-2　软件生命周期的传统的开发阶段

1. 需求分析

软件生命周期的开发阶段从需求分析开始，其主要目标是确定预期系统要提供的服务，这些服务的运行条件（如时间限制、安全性等），以及定义外界与系统的交互方式。

需求分析包括来自预期系统的**利益相关者**（stakeholder，将来的使用者，还有其他有关联的人，比如法律上或者财务上相关的人）提供的重要数据。事实上，如果终端用户是一个实体（如公司或政府机构），他们会为软件项目的实际执行雇用软件开发者，那么需求分析可能开始于用户独自进行的可行性研究。在其他一些情况下，软件开发者可能为大众市场生产COTS（Commercial Off-The-Shelf，商用现成产品）软件，这些软件或许在零售商店销售，或许可通过因特网下载。在这种情况下，用户不再是准确定义的实体，需求分析可能要从软件开发者的市场调研开始。

在任何情况下，需求分析过程都包括：编写和分析软件用户的需求；和项目的利益相关者协商，在一般需求、核心需求、费用和可行性之间权衡；最终确定的需求要明确最终的软件系统必须具有的特性和服务。这些需求被记录在一个称为**软件需求规格说明**（software requirements specification document）的文档中。从某种意义上讲，这个文档是所涉及的各方之间达成的书面协

议，它的目的是指导软件开发，也为日后开发过程中可能产生的分歧提供了解决方法。像IEEE这样的专业组织和美国国防部这样的大型软件客户都已经采用了软件需求规格说明文档编写的标准，这样的事实已经证明软件需求规格说明文档十分重要。

从软件开发者的角度来看，软件需求规格说明文档应该能够为软件的开发顺利进行制定严格的目标。然而，大多数情况下，需求文档很难提供这种稳定性。事实上，软件工程领域里的大多数实践派都认为：在软件工程产业中，导致花费增加和延期交付软件产品的最主要原因是不良沟通以及不断改变客户需求。举例来说，在地基已经建好的情况下，很少有客户会坚持对楼盘的建设计划做大的修改。但是在许多组织机构进行扩编或变更的情况下，软件产品开始构建后，对软件系统的需求也还是会不断涌现（也就是说，软件的需求不会因为软件的构建而停止）。其原因可能是公司决定把原本仅为附属机构开发的软件系统推广到整个公司，或者是技术的进步取代了初始需求分析阶段的可行性。软件工程师已经发现，在任何情况下，与项目的利益相关者进行直接地、经常性地沟通都是必需的。

2．设计

如果说需求分析阶段提供了对一个即将开发的软件产品的描述，那么设计主要是为预期系统的构建提出一个解决方案。从某种意义上讲，需求分析阶段指明要解决的问题，而设计阶段则是制定问题的解决方案。从一个外行人的视角来看，需求分析阶段常常等同于决定软件系统应该做些什么，而设计阶段则是决定系统怎样完成这些目标。虽然这种描述是有意义的，但很多软件工程师认为它是有缺陷的，因为实际上在需求分析阶段有很多要考虑“如何实现”的情况，在设计阶段也有很多要考虑“应该做些什么”的情况。

软件系统的内部结构在设计阶段建立。设计阶段的结果是可被转化为程序的软件系统结构的详细描述。

如果项目是建造一座办公大楼，而不是构建一个软件系统，那么在设计阶段应该为大楼制订详细的结构上的计划并力求满足指定需求。例如，这样的计划应该包含在各个细节层次上描述所建大楼的蓝图汇总。正是源于这些文档，实际的大楼将被建造。制订这些计划的技术已经经历多年的发展，包括标准的符号系统以及大量的建模和图形化方法学。

同样，在软件的设计中画图和建模也发挥着很大的作用。然而，软件工程师所用的方法学和符号系统与建筑领域里所使用的相比，稳定性不太好。确实，与建筑学这个成熟的学科相比，软件工程显得非常动态化，因为软件工程的研究人员一直在努力地寻找软件开发过程中更好的办法。我们将在7.3节探究尚不稳定的软件工程方法，并在7.5节详细讨论当前的符号系统以及与它们相关的图形化/建模方法学。

3．实现

实现阶段涉及程序的具体编写、数据文件的创建和数据库的开发。在实现阶段，我们看到了**软件分析员**（software analyst，有时候也称为系统分析员）和**程序员**（programmer）之间的工作的不同。软件分析员参与整个开发过程，他的工作重点可能在于需求分析与设计步骤，而程序员的主要工作是实现这些步骤。最狭义地说，程序员负责写程序来实现软件分析员提出的设计。做了这样的区分，我们还要注意的是，在计算机领域里并没有一个总的权威来控制术语的使用。许多有着软件分析员头衔的人，本质上就是程序员，而许多有着程序员（也许是高级程序员）头衔的人，从完全意义上讲是软件分析员。我们很快就可以看到，术语上的这种模糊是因为今天的软件开发过程中的步骤经常会交叉重叠。

4．测试

在过去传统的开发阶段中，测试本质上等同于调试程序和确认最终的软件产品是否与软件

需求规格说明文档相一致的过程。但是如今，这样的测试观念被认为太过狭隘。程序不是唯一在软件开发过程中被测试的人工产品，实际上整个开发过程中的每个中间步骤的成果都必须进行精确性测试。此外，我们将在7.6节中看到，现在测试被认为是为全面保证质量所作努力中的一部分，这一目标渗透于整个软件生命周期。因此，很多软件工程师认为测试不应该再被看做是软件开发过程中独立的一步，而是（许多的事例表明）应该纳入到其他步骤中，形成3步开发过程，其中每一步都应该有自己的名称，如需求分析和确认、设计和验证以及实现和测试。

遗憾的是，虽然有现代的质量保障技术，大型的软件系统即使经过了严格的测试，还是可能包含大量的错误，其中许多错误可能在软件的生命周期内都检测不出来。然而，另一些错误可能会造成重大的故障。消除这种错误是软件工程的目标之一。事实上这些方法仍然流行意味着还有许多研究要做。

问题与练习

1. 软件生命周期的开发阶段是如何影响维护阶段的？
2. 简要说明软件生命周期之开发阶段的4个步骤（需求分析、设计、实现和测试）。
3. 试简述软件需求规格说明文档的作用。

7.3　软件工程方法

软件工程早期的方法强调以一个严格的顺序，按照需求分析、设计、实现和测试分阶段进行。理由是，在大型软件的开发过程中，允许做出随意变更会冒太大的风险。结果，软件工程师坚持：在设计之前必须先完成整个系统的需求分析；同样，设计完成后再开始实现。结果产生了现在称为**瀑布模型**（waterfall model）的一个软件开发过程，因为这种开发过程只会按照一个方向进行。

近年来，由瀑布模型规定的高度结构化环境与“自由发挥”的“摸着石头过河”的开发过程之间的矛盾带来了软件工程技术的变化，而后者通常对创造性的问题求解至关重要。软件开发过程中出现的**增量模型**（incremental model）就说明了这一点。依据这个模型，所需的软件系统以一种渐近的模式来构建，即软件产品先是以功能有限的简化版本出现，一旦这个版本的系统通过测试或经未来用户的评估，更多的功能就以递增的方式加到系统中，然后再测试，直至整个系统全部完成。例如，为医院开发的病人记录系统，一开始系统只需要能够查看整个记录系统中的一小部分病人的记录样本就可以了，一旦这个版本的系统能够工作，其他功能（如增加和更新记录的功能等）就可以以渐进的方式加入到系统中去。

另外一种与严格遵循瀑布模型不同的是**迭代模型**（iterative model）。事实上，尽管它与增量模型是不同的，但二者是非常相似（有时是相同）的。增量模型使用扩展产品的每个前期版本到更大版本的概念，而迭代模型则使用改进每个版本的概念。实际上，增量模型通常会包含一个基本的迭代过程，而迭代模型常常渐进增加特性。

一个典型的迭代技术的例子是Rational软件公司创造的RUP（Rational Unified Process，统一软件开发过程），RUP与“cup”押韵，现在这家公司是IBM的一个分公司。RUP在本质上是一种软件开发范型，它重新定义了软件生命周期中开发阶段的每一个步骤，并提供执行这些步骤的指导。这些指导，连同支持它们的CASE工具，都被IBM在市场上交易。今天，RUP在软件领域被广泛地采用。事实上，它的流行促进了非专利版本——**统一过程**（unified process）——的发展，这在非商业基础上非常有用。

增量模型和迭代模型反映出软件开发采用**原型开发**（prototyping）这样一种趋势，也就是把预期系统先做成一个非完整版本，称为**原型**（prototype），并加以评估。在增量模型中，将这些原型发展为一个最终的完整系统的过程称为**演化式原型开发**（evolutionary prototyping）。在迭代性更强的情况中，原型可能会弃而不用，以使得最后设计有全新的实现，这种方法称为**抛弃式原型开发**（throwaway prototyping）。**快速原型开发**（rapid prototyping）通常属于抛弃式原型开发这个范畴，这种方法中，开发过程的早期就很快构建一个预期系统的简单原型。这个原型可能只由几个屏幕图像构成，用来演示系统将如何与用户交互以及系统将有哪些功能。其目标不是作出一个可运行版本的系统，而是作为一个示范工具，来理清软件开发过程中的各个部分相互交流的关系。例如，快速原型有利于在需求分析阶段确定系统需求，也能帮助在销售期间向潜在的客户进行推销介绍。

由计算机的热心者/爱好者使用多年的增量和迭代开发的一种变种方法，称为**开放源码开发**（open-source development）。这是今天许多自由软件开发采用的一种方式。最著名的例子也许就是Linux操作系统，该系统的开放源码开发工作最初是由Linus Torvald领导的。软件包的开放源码开发遵循以下过程。先是单个作者开发一个初始版本的软件（通常是用于满足该作者自己的需求），然后将其源代码和相关文档发放到因特网上，其他用户可以免费下载和使用这个软件。由于这些“其他用户”拥有该软件的源代码和相关文档，他们就能修改或增强这个软件的功能，以使之满足自己的需要，或者是改正他们发现的错误。接下来，他们就将这些改动报告给原作者，原作者就将这些改动整合到自己发布的软件中，使软件的扩展版本可用于更进一步的修改。实际上，一个星期内软件包就有可能经过几次的扩展升级。

由瀑布模型转化而来的最显著的方法就是称为**敏捷方法**（agile method）的方法学集合，它们都建议在增量基础上的早期和快速实现，响应需求变更，降低严格需求分析和设计的重要性。敏捷方法的一个例子就是**极限编程**（XP）。根据XP模型，由少于12名成员组成一个软件开发团队，他们在共同的工作场所自由地交换想法，在开发项目过程中相互协作，通过每天不断重复非正式需求分析、设计、实现和测试这样一个周期开发过程，以增量的方式开发软件。这样，软件包的新扩展版本定期出现，每个新版本都能由项目的利益相关者进行评估，并以此为基础做进一步的增量。概括说来，敏捷方法具有灵活性的特点，这与瀑布模型完全相反。瀑布模型的典型情况就是经理和程序员在各自的办公室工作，并严格地完成整个软件开发任务中明确定义的那部分工作。

比较瀑布模型与XP模型中所描述的差异，揭示了软件工程方法学的广度。这些方法应用于软件开发的过程，期待以一种有效的方式找到更好的方法来构建可靠的软件。这个领域的研究仍在继续，虽然取得了一定的进展，但还有许多工作要做。

问题与练习

1. 概述软件开发的传统瀑布模型与较新的增量和迭代范型之间的区别。
2. 说出3种与严格遵循瀑布模型不同的开发范型。
3. 传统的演化式原型开发与开放源码开发的方法之间的区别是什么？
4. 对于通过开放源码方法开发的软件的所有权而言，你认为可能会出现什么样的潜在问题？

7.4 模块化

7.2节中有一句关键性的陈述：要修改软件，就必须理解这个程序，或者至少是这个程序中

相关的那部分。即使是小程序，要想达到这样的理解也是相当困难的，而对于大型的软件系统，如果没有模块化，那几乎是不可能的。**模块化**（modularity），就是把软件分割成多个易于处理的单元，通常称为**模块**（module），每个模块仅仅承担整个软件的一部分功能。

7.4.1 模块式实现

模块可以以不同的方式实现。我们已经看到（第5章和第6章），在命令型范型的环境中，模块表现为过程。与之对应的是，面向对象范型则是利用对象作为其基本的模块要素。这些差别非常重要，因为它们决定了最初的软件设计过程中的潜在目标。这个目标是将全部工作表示为个别的、易于管理的过程，还是确定系统中的对象并理解它们之间如何相互作用？

为了说明这一点，我们来考虑用命令型范型和面向对象范型是如何开发一个模拟网球比赛的简单模块化程序的。在命令型范型中，我们首先考虑的是肯定会发生的动作。因为每场网球比赛都是从一名选手发球开始，所以我们可以首先考虑构造名为 Serve 的过程（这是基于选手的特性，也许是一个概率点），用来计算球的初始速度和方向。接下来，我们需要确定球的路径。（是否将撞在网上？它将弹回到什么地方？）我们可以把这些计算放在另外一个名为 ComputePath 的过程中。下一步可能就要确定另外一名选手是否能击回这个球。如果能够击回这个球，我们还必须计算球的新的速度和方向，可以把这些计算放在名为 Return 的过程中。

照这样继续，我们可以构造出如图7-3所示的**结构图**（structure chart）所描述的模块化结构。在这个图中，过程用矩形表示，过程之间的依赖关系（由过程调用来实现）用箭头表示。特别是，这个图表明了整个比赛是由名为 ControlGame 的一个过程来控制的。为了完成工作，ControlGame 过程又调用了 Serve、Return、ComputePath 和 UpdateScore 这4个过程的服务。

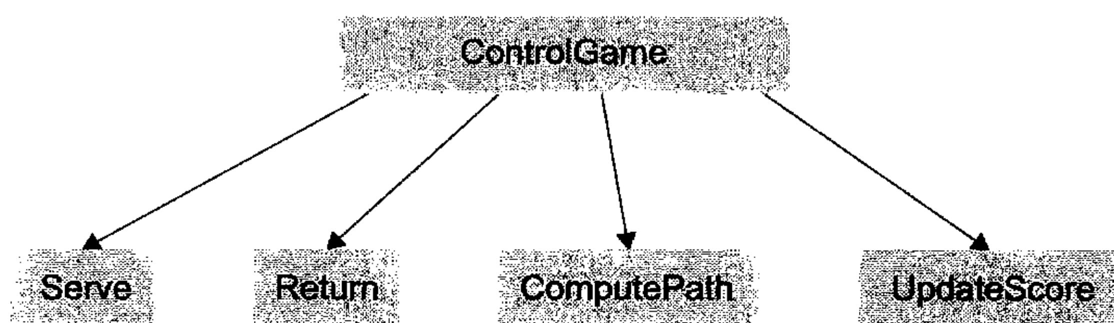


图7-3　一个简单的结构图

注意，这个结构图中并没有描述每个过程如何完成自己的工作，确切地说，这个图仅仅是确定了过程并描述了过程之间的依赖关系。事实上，ControlGame过程要完成自己的工作会先调用Serve过程，然后重复调用ComputePath过程和Return过程，直到有一名选手没有击中球为止。最后，ControlGame在调用Serve过程再重复以上整个过程之前，调用UpdateScore这个过程的服务来更新比分。

至此，我们仅仅是获得了所需系统的一个框架，但思路已经建立起来了。按照命令型范型，通过构思系统必须实现的功能，我们已经完成了程序的设计并得到了设计方案，其中的模块就是过程。

现在，我们重新考虑这个程序的设计，而这次是在面向对象范型的环境中考虑的。我们开始的想法就是用两个对象来表示两位选手，即PlayerA和PlayerB。这些对象将有同样的功能和不同的属性。（两名选手应该都能发球和回击球，但是其技巧和力度不同。）因此，这些对象就是同一个类的实例。注意，我们在第6章介绍了类的概念，类是定义与每个对象相关联的过程（即方法）和属性（即实例变量）的模板。我们把这个类称为PlayerClass，该类将包含Serve方法和Return方法，用来模拟选手的相应动作。这个类中还将包括选手的内部属性（如skill

和endurance等），这些属性的值反映了选手的特征。到目前为止，可以用图7-4来表示我们的设计结果。从图中可以看出，PlayerA和PlayerB是PlayerClass类的两个实例，而这个类包含了Skill属性和Endurance属性，同时也包含了serve方法和returnVolley方法。（注意，在图7-4中我们已经用下划线标注出对象的名称，以此来区分它们与类的名称。）

接下来，我们需要一个对象来实现裁判的功能，帮助判定选手完成的动作是否合乎规则。例如，发球是否过网？球是否落在了球场的合适位置内？为此，我们可以建立一个名为Judge的对象，该对象包含evaluateServe方法和evaluateReturn方法。如果Judge对象判定发球或回球合乎规则，那么比赛继续；否则，Judge对象会给一个名为Score的对象发消息，告之它记录下相应的结果。

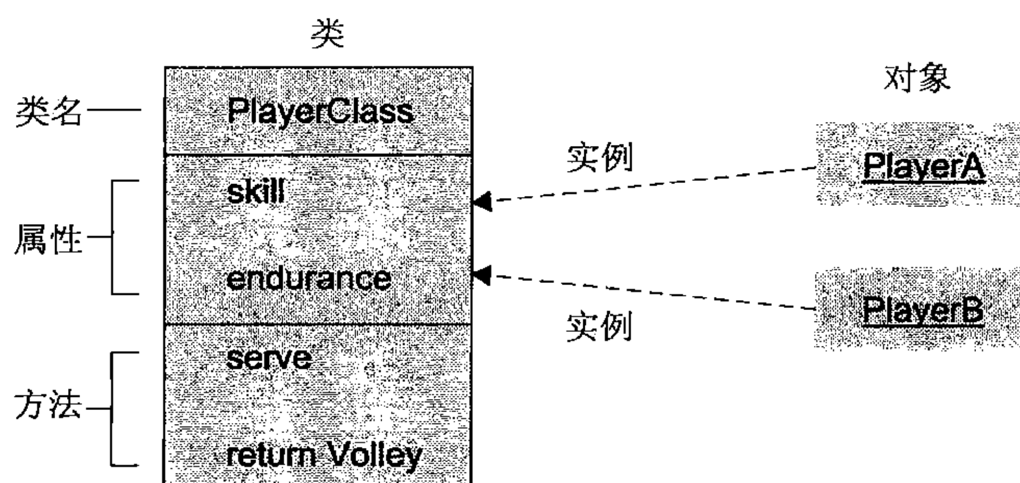


图7-4 PlayerClass类的结构和它的实例

此时，网球程序的设计包括4个对象：PlayerA、PlayerB、Judge和Score。为了说明我们的设计，考虑在网球比赛中可能发生的事件序列（如图7-5所示，图中对象以方框的形式来表示）。这个图是要把对象之间的通信表示成调用对象PlayerA中的serve方法的结果。当我们从上向下依次看图时，会发现事件是按次序发生的。就如同第一个水平箭头所表示的那样，PlayerA通过调用evaluateServe方法向对象Judge报告它的发球，然后对象Judge判定发球是否有效，并且通过调用PlayerB的returnVolley方法请求PlayerB回球。当Judge判定PlayerA产生错误，并且请求Score对象记录下结果时，网球比赛结束。

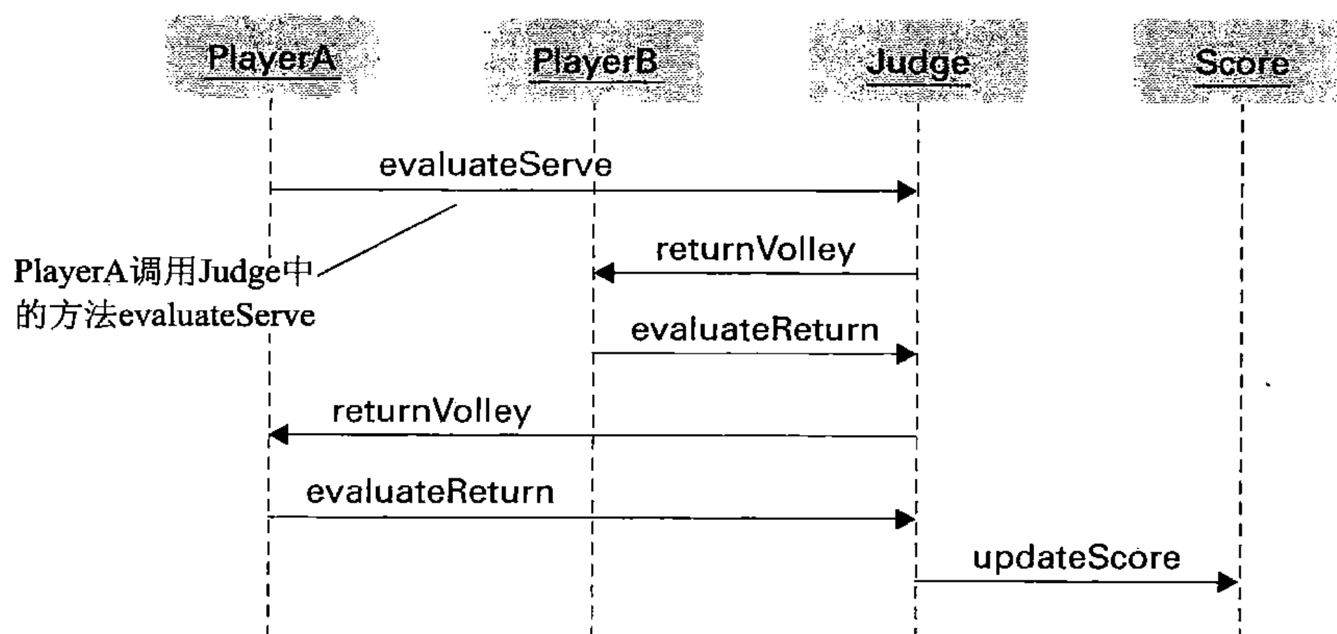


图7-5 由PlayerA的Serve导致的对象间的交互

和命令型范型例子的情况一样，面向对象程序现阶段也是非常简单的。然而，我们已经取得了很大的进步，能够很清楚地理解面向对象模式下是如何进行模块化设计的，而在此模块化设计中，其基本的构件就是对象。

7.4.2 耦合

通过前面的介绍我们知道，模块化是开发出易于管理的软件的一条途径。其基本思想是，以后的任何修改可能只会涉及少数几个模块，允许个人对系统的修改只集中在系统的有关部分，而不是整个系统。当然，这里有个前提，就是对一个模块的修改不会无意中影响到系统的其他模块。因此，当设计一个模块化系统的时候，其目标就应该是做到模块之间的最大独立性，或者换句话说，就是使模块之间的联系尽可能少。这种联系称为模块之间的**耦合**（coupling）。事实上，用来衡量软件系统复杂度（并且这样就获得了一种估算维护软件系统的所需开销的方法）的一个指标就是度量该系统的模块间的耦合。

模块间的耦合有多种形式。一种是**控制耦合**（control coupling），出现在一个模块移交执行控制给另外一个模块时，如过程调用的情况。图7-3里的结构图就表示了存在于过程之间的控制耦合。具体来说，从ControlGame模块到Serve模块之间的箭头说明了前者将控制权传递给后者。图7-5中的结构图也代表了一个控制耦合的情况，图中的箭头所描绘的路径就代表了控制权在对象之间的传递。

模块间的另一种形式的耦合是**数据耦合**（data coupling），这是指模块间的数据共享。如果两个模块是通过共享同一个数据项相互作用的，那么当对一个模块进行修改时，可能会影响到另外一个模块，并且对数据本身格式的修改在这两个模块中都会有反映。

过程间的数据耦合有两种形式。一种是以参数的形式从一个过程到另一个过程进行显式的数据传送。这种耦合在结构图中是这样表示的，即用过程之间的箭头指示数据的传送。箭头的方向表明在此方向上进行数据项的传送。例如，图7-6是图7-3的扩展版本，在此图中，我们可以看出当ControlGame过程调用Serve过程时，ControlGame过程会将需要模拟的那位选手的属性告知给Serve过程。当Serve过程完成后，它就将球的轨迹报告给ControlGame过程。

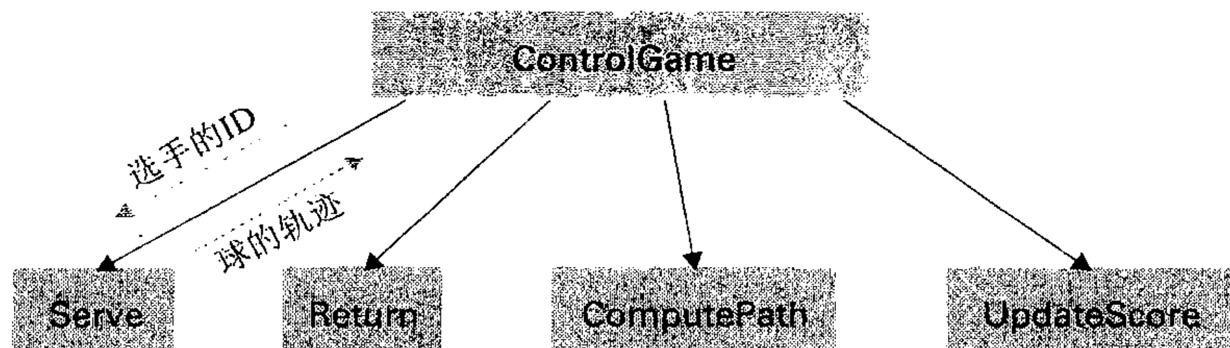


图7-6　包含数据耦合的一个结构图

类似的数据耦合也发生在面向对象设计中的对象之间。例如，当PlayerA对象请求Judge对象对其发球进行判定时（见图7-5），它必须将球的轨迹信息传递给Judge对象。另一方面，面向对象设计模式的一个优势就在于它从本质上倾向于将对象之间的数据耦合减小到最低。这是因为对象的方法易于包括操作对象内部数据的所有过程。例如，PlayerA对象将包括该对象的有关属性信息和针对这些信息的处理方法。因此，没有必要将这些信息传递给另外一些对象，这样对象之间的数据耦合就能达到最小。

与通过参数进行显式的数据传递方式不同的是，数据可以以**全局数据**（global data）的形式在模块之间进行隐式共享。全局数据是可以自动地被整个系统中的所有模块使用的数据项。这与局部数据项不同，局部数据项只能在某个特定的模块中使用，除非明确地传递给了另外一个模块。大多数高级语言提供了全局数据和局部数据的实现方法，但是对全局数据的使用应当谨慎。全局数据使用的问题在于，如果某个人试图修改依赖于全局数据的一个模块，那么他就很难确定正被修改的模块与其他模块之间有怎样的相互关系。简而言之，全局数据的使用降低了模块作为一种抽象工具的使用价值。

7.4.3 内聚

正如模块间的耦合应最小化，同样重要的是，每个模块的内部绑定程度应该最大化。术语**内聚**（cohesion）就用来表示这种内部绑定，或者说模块内部各部分的关联程度。为充分理解内聚的重要性，必须考察系统的最初开发并考虑这个软件的整个生命周期。如果有必要在模块中作出修改，那么存在于模块中的各种不同的活动会搅乱原本简单的一个过程。所以，软件设计人员在寻求模块间的低耦合的同时，还力求做到模块内部的高内聚。

一种内聚度较弱的内聚形式称为**逻辑内聚**（logical cohesion）。模块内的逻辑内聚是由其内部元素本质上实现逻辑上相似的活动所引起的。例如，考虑一个模块，它完成整个系统与外界进行通信的功能。粘合这个模块的"胶水"是模块中的所有活动都用以处理通信。然而，通信的主题各不相同，有的可能是用来获取数据，有的可能是用来报告结果。

一种内聚度较强的内聚形式称为**功能内聚**（functional cohesion）。这就表示模块中所有部分都集中于实现某一项功能。在命令型范型的设计中，如果把模块的子任务独立在其他模块中，并将这些模块用作抽象工具，那么该模块的功能内聚的程度通常会增强。这一点在模拟网球比赛的例子中得到了很好的说明（参见图7-3）。在该图中`ControlGame`模块将其他模块用作抽象工具，以便它能集中调度整个比赛，而不是把精力分散在实现发球、回击球和维护比分这样的细节上。

在面向对象设计中，因为对象中的方法常常执行松散相关的活动，其唯一的共同"纽带"就是它们都是由同一个对象执行的活动，所以全部的对象通常在逻辑上内聚。例如，在模拟网球比赛的例子中，每个选手对象都包含发球和回击球的方法，这些方法是明显不同的活动，所以这样一个对象仅仅是在逻辑上内聚的模块。然而，软件设计人员应当力求做到使一个对象中的每个方法都在功能上内聚。也就是说，即使对象在整体上仅仅是逻辑上内聚，对象里的每个方法也应当只实现一个功能内聚的任务（参见图7-7）。

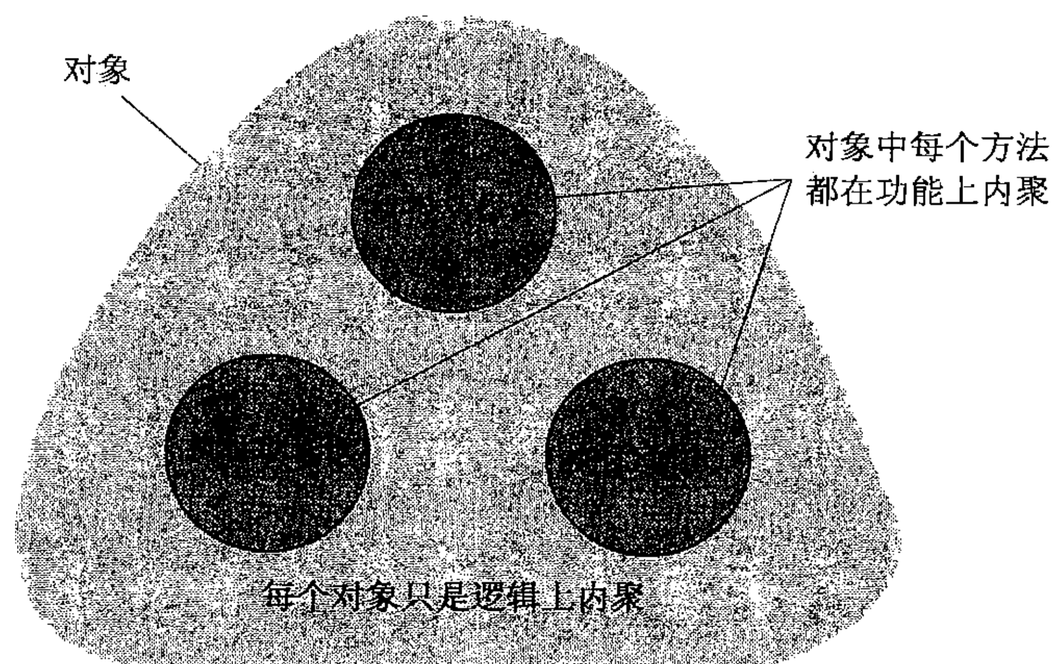


图7-7 一个对象的逻辑内聚和功能内聚

7.4.4 信息隐藏

信息隐藏（information hiding）是好的模块化设计的一个基本特征，它指的是限制软件系统的指定部分的信息。这里的术语信息应该从广义阐释，它包括关于程序单元结构和内容的任何知识。因此，它包括数据、用到的数据结构类型、编码系统、模块的内部组成结构、过程单元的逻辑结构和任何涉及模块内部特性的因素。

信息隐藏的关键就是阻止模块的动作对其他模块产生不必要的依赖或影响。否则，就有可能导致模块产生错误，这些错误可能是在其他模块的开发中带来的，亦或是在软件维护期间不正确的维护带来的。例如，如果一个模块不限制其他模块对其内部数据的使用，那么这些数据可能就会被其他模块破坏。或者，如果设计一个模块以利用另一个模块的内部结构，如果其内部结构被修改了，随后它将可能产生错误。

要注意到信息隐藏具有两个化身，这是非常重要的。一个是作为设计目标的，另一个是作为实现目标的。应当这样设计一个模块，使其他模块不需要读取它的内部信息，并且应当以强化模块边界的方式实现一个模块。前者的例子是最大化内聚和最小化耦合。后者的例子涉及使用局部变量、应用封装和使用完善定义的控制结构。

最后，我们应该注意到信息隐藏对于抽象主题和抽象工具的使用极为重要。实际上，抽象工具的概念是“黑盒”的概念，用户可以忽略它的内部特性，这样就允许用户集中考虑手头更大的应用。在这种情况下，信息隐藏相当于封装抽象工具的概念，就像安全罩可以用来保护复杂的、具有潜在风险的电子设备一样。保护他们的用户远离内部危险，同样也保护内部，以防来自其他用户的侵扰。

7.4.5　构件

我们已经提到，软件工程领域里的一个障碍就是缺乏预制的现成构件块来构建大型的软件系统。在这一点上，软件开发中的模块化方法让我们看到了希望。特别是，面向对象程序设计范型显得尤其有用。这是因为对象，它们来自完备的、自我包含的单元，这些单元明确定义了与其外部环境的接口。一旦对象（更准备地说，是一个类）设计成能完成某种特定功能时，它就可以在任何要求提供这种服务的程序中用来实现这个功能。此外，继承提供了一种对预制对象的定义进行改进的方法（在对象定义必须定制以符合一个特定应用的需要时）。

于是，面向对象编程语言C++、Java以及C#都伴随有一组预制的“模板”这一点就不足为奇了。通过这些模板，程序员可以很方便地实现对象并用来完成特定功能。具体来说，C++拥有C++标准模板库，Java编程环境伴随有Java应用编程接口（API），C#程序员可以访问.NET框架类库。

对象和类有可能为软件设计提供预制的构建块，但是它们还不太完美。一个问题是它们提供相对较小的模块来构建系统，所以对象实际上是更通用的**构件**（component）概念中的一个特例，构件就是软件的一个可复用单元。实际上，大多数构件都是基于面向对象范型的，并且表现为一个或多个对象组成的集合的形式，其功能是作为一个自包含单元。

构件的开发和利用的研究导致了称为**构件架构**（component architecture，也就是通常所说的基于构件的软件工程）的领域的出现。在此领域中，传统的程序员被**构件装配员**（component assembler）所代替，由构件装配员把预制的构件装配成软件系统。在许多开发环境中，常常用图形界面中的图标来表示预制的构件。构件装配员并不涉及构件内部的编程，而是在预先定义好的构件集合中选择相关的构件，然后将它们进行最小化的定制并连接，从而获得所需要的功能。确实，一个设计好的构件的属性就是不需要经过内部的修改就可以进行扩展，来包含一些针对特定应用的特性。

构件架构在智能手机系统这一领域中尚无用武之地。因为这些设备的资源有限，数个应用实际上只是一组相互协作的构件的集合，每个构件会谨慎地为其应用提供某个功能。例如，一个应用中的每个显示屏通常是一个独立的构件。在其背后，可能存在其他服务构件，或用于存储和访问存储卡上的信息，或执行某个持续的功能（如播放音乐），或用于通过因特网访问信息。

每一个这样的构件都按需独立启动和终止，这样才能高效地服务于用户。但是，应用本身看起来却是显示和动作的一个无缝系列。

现实世界中的软件工程

以下情节是现实世界的软件工程师所面临的典型问题。XYZ公司聘请一家软件工程公司为其开发、安装一套全公司的集成软件系统，以满足公司的数据处理需要。作为系统的一部分，XYZ公司又建立了一套PC机网络系统，让员工用来访问这个全公司系统。这样，每个员工的办公桌上都有一台PC机。很快这些PC机不仅能用来访问新的数据管理系统，而且可以作为可定制的工具，让员工用其来提高自己的工作产出。例如，某名员工可以开发一个电子制表程序来提高工作效率。遗憾的是，这样一个定制的应用可能设计并不完善，或者没有经过彻底的测试，而且可能会涉及这名员工并不能完全理解的一些特性。随着年限的增加，这些定制的应用程序慢慢会融合到公司的内部事务处理过程中。与此同时，当初开发这些应用程序的员工可能会升迁、调任或者会离开这家公司，而使用这些程序的其他同事却并不懂这些程序。结果，起初的一个精心设计、协调一致的系统变成了一个设计差、无文档的、出错频繁的拼凑品。

问题与练习

1. 小说与百科全书在其单元（如章、节以及条目）之间在耦合程度方面有什么不同？内聚方面呢？
2. 一项体育活动通常划分为一些单元。例如，棒球比赛被分成了几个回合，网球比赛被分成了几局。试分析这些“模块”之间的耦合性。这些单元的内聚到了怎样的程度？
3. 最高程度的内聚与最小程度的耦合的目标是否一致？也就是说，随着内聚度的增加，耦合度会相应降低吗？
4. 定义耦合、内聚和信息隐藏。
5. 扩展图7-3中的结构图，使其包括`ControlGame`与`UpdateScore`模块之间的数据耦合。
6. 如果`PlayerA`的发球因违反比赛规则被视为无效，绘制一幅类似于图7-5的图，表示发生的事件序列。
7. 传统的程序员与构件装配员之间有什么区别？
8. 假设大多数智能手机都有一些个人组织应用（如日历、联系人、闹钟、社会化网络、电子邮件系统、地图等），你认为构件功能的哪些组合实用而有趣呢？

7.5 行业工具

本节里，我们研究一些在软件开发的分析与设计阶段使用的建模技术和符号系统。其中一些技术和符号系统是在软件工程学科中以命令型范型为主导的年代里开发的。现在，在面向对象范型环境中也可以找到它们中某些的身影，而另外的一些如结构图（见图 7-3）则是专门用于命令型范型的。我们首先考虑一些从命令型范型发展而来的技术，然后研究较新的面向对象的工具和设计模式的扩展功能。

7.5.1 较老的工具

尽管命令型范型致力于依据过程来构建软件，但确定这些过程的方法是考虑将被操作的数据，而不是过程本身。其思路是，研究数据在系统中如何流动，就能确定在哪儿修改数据格式，或者在哪儿对数据的路径进行合并或拆分。因此，就确定了进行处理的位置，这样一来，通过

数据流的分析就能确定过程。**数据流图**（dataflow diagram）是表示从数据流分析过程中所获得的信息的一种方法。在数据流图中，箭头表示数据路径，椭圆表示数据操控发生的地点，矩形表示数据源和数据存储。作为一个示例，图7-8表示的是医院账单系统的一个基本的数据流图。注意，该图表明Payments（从病人中“流出”的）和PatientRecords（从医院文件中“流出”）在椭圆ProcessPayments处合并，并从此处将UpdatedRecords“流回”到医院文件。

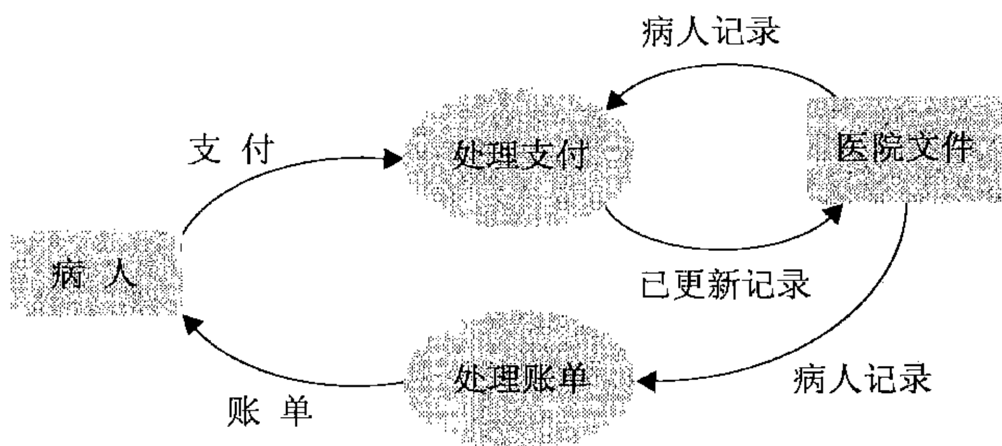


图7-8　一个简单的数据流图

数据流图不仅能在软件开发的设计阶段帮助确定过程，还能在分析阶段帮助理解预期系统。实际上，构建数据流图可以作为一种用来改善客户与软件工程师之间的交流的方法（因为软件工程师一直为理解客户需要什么而努力并且客户努力描述个人愿望），所以，即使在命令型范型已经不太流行的情况下，这些数据流图还有其应用价值。

软件工程师已经用了很多年的另一种工具就是**数据字典**（data dictionary），它是关于整个软件系统中出现的数据项的一个中央信息库。这些信息包括：为引用每个数据项所采用的标识符、每个数据项里的有效条目的构成情况（数据项一直是数字型的，还是一直是字符型的？分配给该数据项的值的可能范围是什么）、数据项存放在什么地方（数据项是要存放在文件中或数据库中吗？如果是这样的，具体存放在哪一个里面）、软件在什么地方会引用这些数据项（哪些模块需要数据项的信息）。

构建数据字典的一个目标是，增强软件系统的潜在利益相关者与软件工程师之间的沟通，其中软件工程师负责将利益相关者的需求转化为需求规格说明文档。在这样的环境下，构建数据字典有助于确保这样一个事实，即如果部分数字不是真正的数字型的，那么在软件的分析阶段就可以发现，而不用等到在后面的设计和实现阶段才发现这个问题。构建数据字典的另一目标是确立整个系统的一致性。借助构建字典常常可发现冗余和矛盾。例如，一个数据项在库存记录中称为 PartNumber，而在销售记录中可能就改称为 PartId。还有，在人事部门可能会用 Name 这个术语来表示一名员工，而在库存记录中可能用来表示一个零件。

7.5.2　统一建模语言

数据流图以及数据字典是在面向对象范型出现以前，软件工程领域发展比较成熟的一些工具。即使是在命令型范型（它们最初是针对命令型范型开发的）现在已经不太流行的情况下，这些工具还是能继续找到其应用价值。现在，我们转而研究更为先进的工具集，称为 UML（Unified Modeling Language，统一建模语言）。统一建模语言是基于面向对象范型思想发展而来的。然而，在这个工具集中，我们讨论的第一个工具是**用例图**（use case diagram）。无论其潜在的范型如何，这个工具都是非常有用的，因为它仅仅尝试着从用户的视角来捕捉预期系统的画面。图 7-9 表示的就是用例图的一个例子。

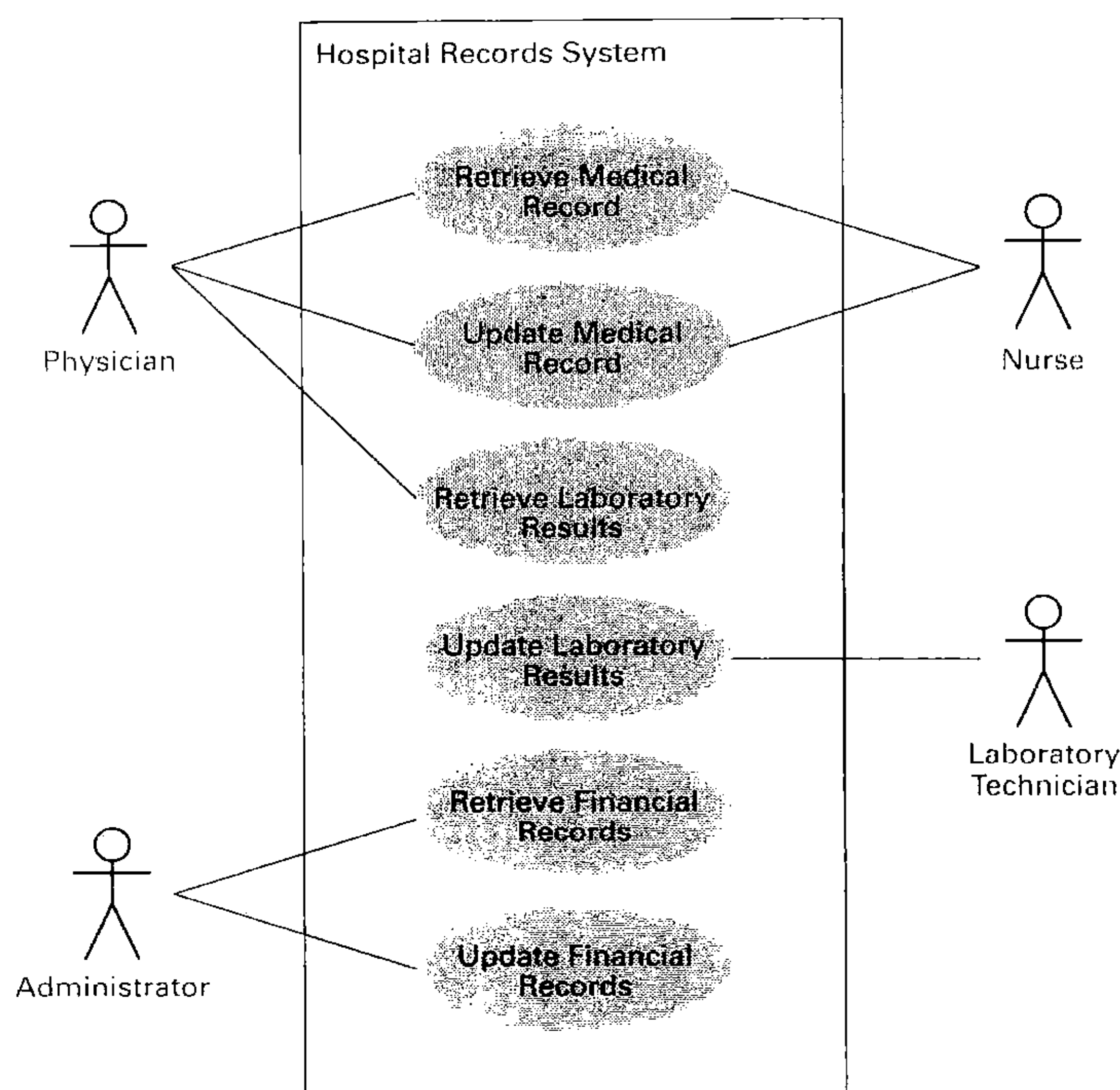


图7-9 一个简单的用例图

用例图是用大的矩形框来描述预期的系统，在这个矩形框中，系统与其用户之间的交互——称为**用例**（use case）——是用椭圆来表示的，而系统中的用户——称为**参与者**（actor）——用火柴人表示（即使角色可能不是一个人，也这么表示）。这样，图7-9所表示的就是Hospital Records System，该系统在获得Physician或Nurse的请求时，就会完成Retrieve Medical Records这个用例。

鉴于用例图是从预期系统的外部来观察系统的，所以UML提供了许多种工具，用于表示系统内部的面向对象设计。其中的一种工具是**类图**（class diagram），它是一个标记系统，用来表示类的结构和类之间的联系——在UML的术语中称为**关联**（association）。举一个例子，考虑医生、病人和病房之间的关系，我们假定表示这些实体的对象是分别从类Physician、Patient和Room构造出来的。

图7-10表明了Physician类、Patient类以及Room类之间的联系在UML类图中是如何表示的。用矩形框表示类，用线来表示关联，关联线上可能有标号，也可能没有。如果有，那粗体箭头就可用来指明标号被读的方向。例如，在图7-10中带标记cares for的箭头指示医生医治病人，而不是病人医治医生。有时关联线上带有两个术语标记，可以从任一方向读取关联。图7-10中的类Patient和Room之间的关联就印证了这一点。

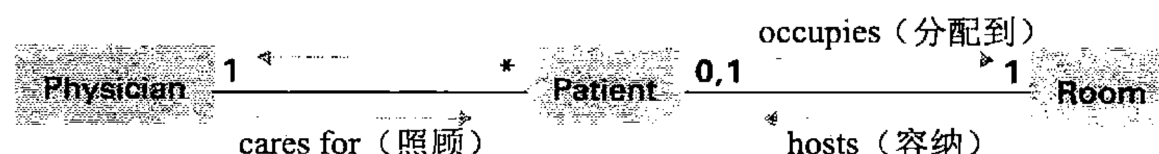


图7-10 一个简单的类图

除了指明类之间的关联之外，类图还能表达这些关联的多样性。也就是说，它能指明一个类的多少个实例与另一个类的实例相关联。这个信息被记录在关联线的两端。图7-10指明每位

病人可以占据一个房间，而每个房间能供0位或1位病人住宿。（我们假定每个房间都是私人房间。）*表示一个任意的非负数。这样，图7-10中的*表示每位医生可以医治多位病人，而在关联的医生端的1表示每位病人只被一位医生医治。（我们的设计只考虑主治医生的作用。）

为了完整性起见，我们应该注意到关联的多样性有3种基本形式：一对一联系、一对多联系和多对多联系，如图7-11所示。**一对一联系**（one-to-one relationship）的一个例子就是病人和私人病房之间的关系，其中每位病人只能分配到一个房间，而且每个病房只分配给一位病人。**一对多联系**（one-to-many relationship）的一个例子就是医生和病人之间的关系，其中每位医生可以照顾多位病人，而每位病人只有一位（主治的）医生照顾。在这个例子中，当我们考虑将病人与咨询医生之间的联系加入病人与医生之间的联系时，就形成了**多对多联系**（many-to-many relationship），即每位病人可以有多位咨询医生来辅助治疗，而每位咨询医生可以帮助多个病人。

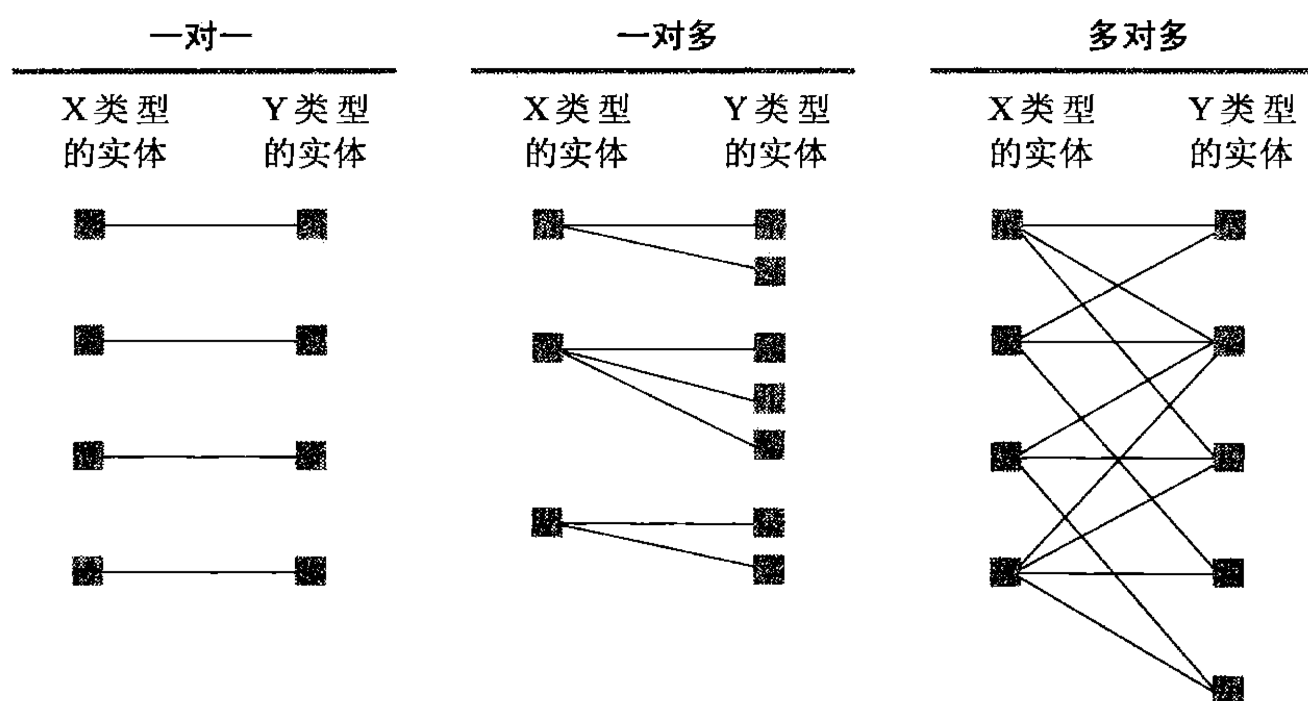


图7-11　X类型实体与Y类型实体间的一对一、一对多以及多对多联系

在面向对象的设计中，经常会出现一个类表示另一个类的更加具体的版本。在这种情况下，我们说后者是前者的泛化。UML提供了特殊的符号来表示泛化。图7-12给出了一个例子，它描述了类MedicalRecord、SurgicalRecord和OfficeVisitRecord间的泛化。类间的关联用带空箭头的箭头表示，这是UML表示泛化的关联符号。注意，每一个类都是由一个矩形表示，里面包含了类的名称、属性和方法（格式参见图7-4）。这是UML在类图中表示类的内在特征的方法。图7-12中描述的信息是：MedicalRecord类是SurgicalRecord类的泛化，同时也是OfficeVisitRecord类的泛化。也就是说，SurgicalRecord类和OfficeVisitRecord类包含了MedicalRecord类的所有特征，并附加了那些明确地列在它们矩形框中的特征。因此，SurgicalRecord类和OfficeVisitRecord类都包含病人的姓名、医生的姓名和病历日期，但SurgicalRecord类还包含手术流程、医院、出院日期和准许病人出院的权力，而OfficeVisitRecord类包含了症状和诊断。这3个类都有打印医疗记录的功能。SurgicalRecord类和OfficeVisitRecord类中的PrintRecord方法是MedicalRecord类中PrintRecord方法的特殊化，它们都可以打印其类特有的信息。

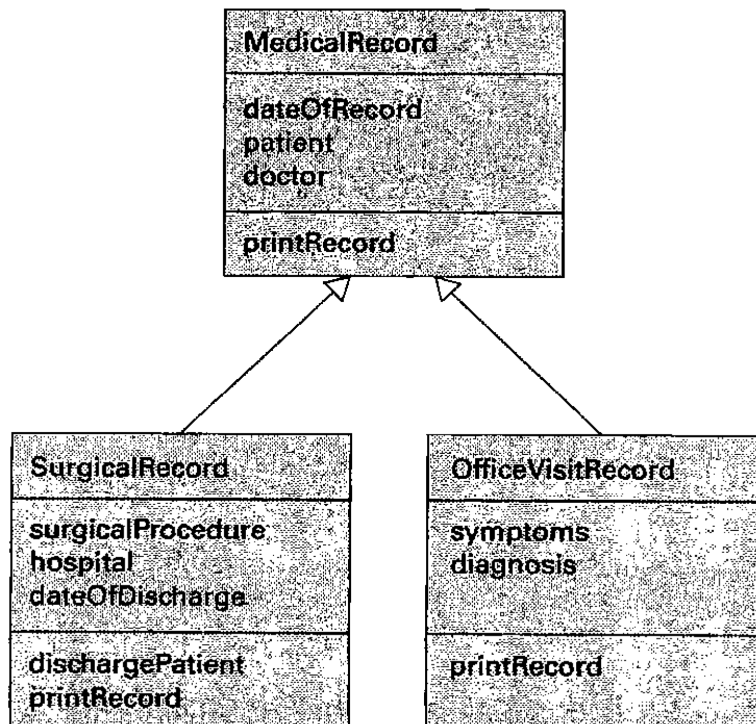


图7-12　描述泛化的一个类图

回顾第6章（6.5节），在面向对象编程环境中实现泛化的一个很自然的方式就是利用继承。然而，许多软件工程师都告诫说，继承并不是对所有的泛化情况都适合。原因在于，继承导致了类间的强耦合度，这种耦合在软件生命周期的后期并不希望出现。例如，类的改变会自动地在它的所有继承类中得到反映，因此在软件维护阶段看起来很小的改动就能够导致不可预见的后果。作为一个例子，我们可以假设一个公司为其员工开放一个娱乐设施，这也就意味着娱乐设施里的所有具有成员资格的人员都是该公司的员工。为了给这个设施做一个成员表，程序员可以利用继承依据早先已经定义的`Employee`类构建一个`RecreationMember`类。但是，如果随着公司后来的效益提高，公司决定对员工的家属和退休员工也开放娱乐设施，于是，`Employee`类和`RecreationMember`类之间内含的耦合性问题将会变得更为严重。所以，使用继承的时候不应当只考虑其方便性，而应当将继承的使用严格限制在需要实现的泛化一直不会更改的情况下。

类图代表的是程序设计中的静态特征，它不能表示程序在执行过程中发生的事件序列。为了表示这种动态特征，UML提供了一系列图的类型，它们统称为**交互图**（interaction diagram）。交互图的一种是**序列图**（sequence diagram），它描述了完成任务所涉及的个体（如参与者、完整的软件构件或个体对象等）间的通信。这些图与图7-5类似，因为它们都用带有向下延伸的虚线的矩形表示个体。每个矩形连同它的虚线称为**生命线**（life line）。个体间的通信用连接合适生命线的带标记的箭头表示，这里的标记指示被请求的动作。当自顶向下阅读图时，这些箭头是按时间先后次序出现的。当个体完成请求的任务，并把控制返回给发出请求的个体（就像传统的从一个过程返回）时，这时用一个指回原始生命线的无标记箭头表示通信。

因此，图7-5从本质上讲是一个序列图。但是，图7-5的语法本身有几个缺点。一个就是它不允许获取两对手间的对称，我们必须画出单独的图来表示开始于`PlayerB`发球的网球，即使交互的序列与`PlayerA`发球的非常相似。此外，图7-5只描述了一次具体的击球，一次一般的击球活动肯定可以无限延展。形式化序列图有在单个图中获取这些变化的技术，虽然我们不需要仔细研究这些，但还是应该简要地看一下图7-13中显示的形式化序列图，它描述了基于我们的网球比赛设计的一个一般的击球活动。

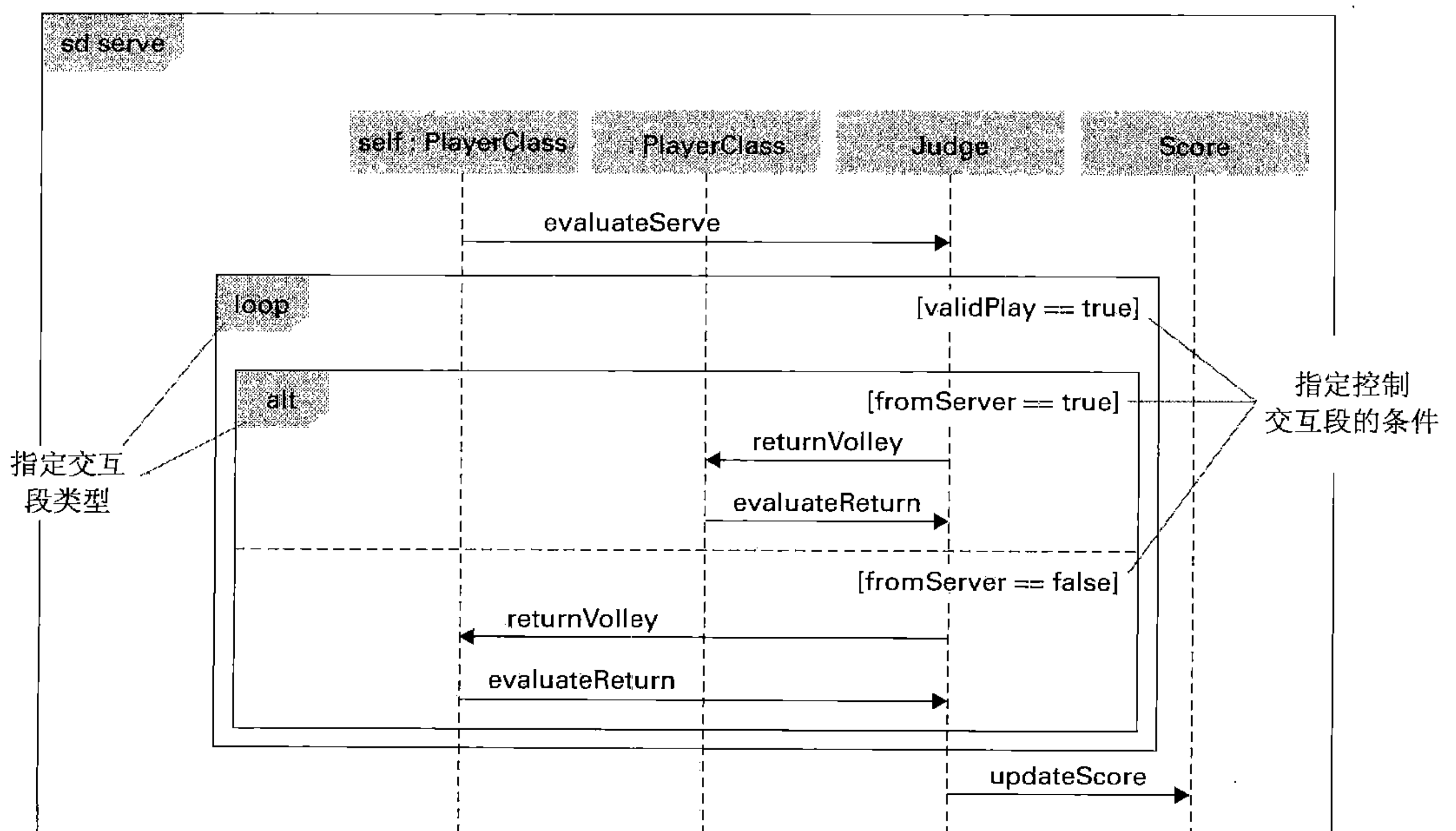


图7-13 描述一般网球的序列图

还要注意图7-13说明了整个序列图是包含在一个矩形——**帧**（frame）——中的。帧的左上角是一个包含了跟有标识符的字符sd（意思是“sequence diagram”）的五角形，这个标识符可能是标记整体序列的名字，或（正如图7-13中的）是被调用来初始化序列的方法的名字。注意，与图7-5对比，图7-13中表示参赛手的矩形并没有指定具体的参赛手，而仅仅指示它们代表`PlayerClass`“类型”的对象。其中一个被指定为self，意思是这是一个其`serve`方法被激活去初始化序列的对象。

关于图7-13的其他关键点是它处理两个内部的矩形，即**交互段**（interaction fragment），它们用来表示一个图中的候选序列。图7-13包含了两个交互段，一个标记为“loop”，另一个标记为“alt”。这本质上是我们首次在5.2节伪代码中遇到的`while`和`if-then-else`结构。“loop”交互段表明边界内的事件将重复，只要`Judge`对象判定`validPlay`的值为真；“alt”交互段表明根据`fromServer`的值是真是假，其中一个候选序列被执行。

最后，在这里介绍**CRC卡**（Class-Responsibility-Collaboration card，类-职责-协作卡）的功能还是比较合适的，尽管这部分内容不属于 UML，但它在验证面向对象设计的有效性方面起着很重要的作用。CRC 卡是一张简单的卡片（如索引卡片），上面写着有关对象的描述。利用这种方法，软件设计师为预期系统的每个对象做一张卡片，然后在模拟系统中用这些卡片来表示对象，可以在桌面上进行，也可以通过一个“舞台表演”的实验，在实验中，设计团队的每个成员手持一张卡片，然后扮演卡片上所描述对象的角色。这样的模拟通常称为**结构化走查**（structured walkthrough），在设计阶段的找错能力要优于设计的实现阶段，因而被证明在设计阶段是一种比较有效的方法。

7.5.3 设计模式

对软件工程师而言，越来越有用的工具是不断发展的设计模式集。**设计模式**（design pattern）是用来解决软件设计过程中反复出现的问题的一种预先开发的方法。例如，**适配器**（Adapter）模式提供了一个解决办法，用来解决通过预制模块来构建软件的过程中经常出现的问题。具体来说，预制模块可能已经具备了解决手边问题的功能，但可能还没有与当前应用相一致的接口。在这样一种情况下，适配器模式可以用作一种标准方法，将模块封装在另外一个模块里，仅仅需要为原始模块的接口与外部世界提供解释功能，这样一来，就允许原始的预制模块用于该应用中。

另一种成熟的设计模式是装饰者（Decorator）模式，它提供了一种用来设计系统的方法，而所设计的系统依据当时的环境完成一些相同的活动的不同组合。这种系统会产生大量的选择，如果没有经过仔细的设计，很可能导致软件极大的复杂度。但是，装饰者模式提供了一个实现这类系统的标准化方式，从而产生了一种易于管理的解决办法。

系统设计的悲剧

对系统完善设计的严格要求可以通过一个例证得到说明，这就是Therac-25（20世纪80年代中期，医学界使用的一台基于计算机技术的电子加速放射治疗仪）所遇到的问题。该机器的设计缺陷导致了6例放射过量的事件发生，其中3例导致人员的死亡。设计的缺陷包括：(1) 机器界面设计的不合理，使得操作员在机器的放射量调整到合适值之前就能进行放射操作，(2) 硬件与软件的设计之间的协作性差，结果就导致了某些安全性能的失效。

在一些更近的例子中，有因设计不当导致大面积停电、电话服务中断、金融业务的重大错误、空间探测器的失踪以及因特网的瘫痪等。如果你想对这个问题了解更多，请查阅风险论坛（它的网址是http://catless.ncl.ac.uk/Risks）。

设计模式中的重复问题的识别以及设计模式的创建和分类在软件工程领域里是一个不断进步的过程。然而，其目标不仅仅是找到设计问题的解决办法，还要找到高质量的解决方案，这种解决方案在软件生命周期的后期能提供很好的灵活性。因此，对诸如耦合最小化和内聚最大化这样好的设计原则的考虑，在设计模式的发展过程中起着重要的作用。

设计模式在发展过程中所取得的进展成果，在今天的软件开发包所提供的工具库中得到了体现，如 Oracle 公司提供的 Java 编程环境以及微软公司提供的.NET 框架等。事实上，在这些“工具包”中找到的大多数“模板”本质上是设计模式的框架，这就为设计问题找到了现成的、高质量的解决方案。

最后我们要提到的是，软件工程里的设计模式的出现是不同的领域相互促进的一个很好的例子。设计模式的起源来自于 Christopher Alexander 在传统建筑领域里的研究，他的目标是发现那些提高建筑设计质量的特征，然后开发包含这些特征的设计模式。今天，软件设计中已经包含了他的许多思想，并且许多软件工程师仍继续从他所做的工作中汲取灵感。

问题与练习

1. 请画一个数据流图，用来表示当一名读者从图书馆借书时的数据流。
2. 请画出图书馆记录系统的用例图。
3. 请画出一个类图，用来表示旅客与他们住的酒店之间的联系。
4. 画出表示人是雇员的泛化的类图，包括可能属于每个类的一些属性。
5. 把图7-5转化为完整的序列图。
6. 在软件工程的过程中，设计模式扮演着什么样的角色？

7.6 质量保证

软件故障、费用超标、逾期等现象的迅速产生，对软件质量控制方法的改进提出了要求。本节我们考虑一些在此方面的努力方向。

7.6.1 质量保证的范围

在计算机技术发展的早期，生产合格软件的关注点主要集中在去除在实现过程中产生的编程错误。在本节的后面，我们将讨论在这方面上取得的进步。然而，如今软件质量控制的范围远超出了调试过程，它的分支包括改进软件工程过程，开设培训课程以确保员工具有上岗资格，以及确立健全的软件工程标准等。在这方面，我们已经注意到像ISO、IEEE和ACM这些组织在提升职业化程度和设立标准（以评估软件开发公司内部的质量控制）方面所起的作用。一个典型的例子是ISO9000系列标准，它提供给许多像设计、生产、安装、服务这样的工业活动。另外一个例子是ISO/IEC 15504，它是由ISO和国际电工委员会（IEC）联合制定的一套标准。

现在大多数软件承包商要求他们雇用来开发软件的组织符合这样的标准。因此，软件开发公司正在建立**软件质量保证（SQA）小组**，它负责监督和强制执行组织采用的质量控制系统。这样，在传统的瀑布模型下，SQA小组将负责在设计阶段之前批准软件需求规格说明，或在实现开始前批准设计及相关的文档。

许多主题构成了当今质量控制的基础，其中之一就是记录保存。为了将来能够做参考，在开发过程中的每一步都要被准确地记入文档，这是非常重要的。但是，这个目标与人类的本性相冲突，人的本性使得人们倾向于在作出决定或改变决定时不修改相关文档。因此，记录有可

能是不正确的，从而在未来阶段使用它时很有可能会产生严重后果。在这一方面，CASE工具具有非常大的好处，它使得像重画示意图和更新数据字典这类任务与手工方法相比，要更加容易。因此，记录更可能会被更新，最终的文档也更可能是准确的。（这只是软件工程必须应对人性弱点的多个实例之一，其他的例子包括当人们共事时，不可避免的人性冲突、嫉妒、自我抵触等。）

另一个与质量相关的主题是**评审**（review）的使用，其中涉及软件开发项目的各方聚在一起，考虑一个指定的话题。评审贯穿整个软件开发过程，采用的形式是：需求评审、设计评审和实现评审。在需求分析的早期，评审可能表现为原型演示，或为软件设计团队成员间的结构化走查，或为实现设计相关部分的程序员间的协调。这样的评审（基于重复的基础）提供了沟通的渠道，通过它误解得以避免，错误在造成灾难前得以更正。评审的重要性已经被这样的事实佐证：在IEEE标准中，对于软件评审有专门的论述，这就是众所周知的IEEE1028。

有些评审在本质上是关键的。一个例子就是项目利益相关者的代表和软件开发团队之间进行的评审，用以批准最终软件需求规格说明文档。实际上，获批就标志了需求分析阶段的正式结束，同时它也是后续开发过程进行的基础。但是，从质量控制的角度来说，所有的评审都是重要的，它们都应该记入文档，作为不断进行的记录维护过程的一部分。

7.6.2 软件测试

软件质量保证现在被认为是贯穿整个开发过程的一个主题，程序的测试和验证本身一直是研究的主题。在5.6节中，我们讨论了用数学上严格的方法验证算法正确性的技术，但结论是如今大多数软件要使用测试来"验证"。但是，这种测试最多只是一种不精确的方法。除非我们对一个软件做足够多的测试，穷尽所有可能的情况，否则还是不能保证这个软件没有错误。即使是简单的程序，也可能有无数条可以遍历的路径。所以，对一个复杂的程序的所有可能的路径进行测试是不可能的。

另一方面，软件工程师已经开发出了一些测试方法，在经过有限次测试的情况下，可提高发现软件错误的可能性。其中一种是基于这样的观察，即软件中的错误趋于类聚。也就是说，经验表明，一个大型的软件系统中会有一小部分模块比其他模块更容易出问题。所以，与其把所有的模块都进行相同的、不彻底的测试，还不如确定那些容易出错的模块，对它们进行彻底的测试，这样可以发现系统的更多错误。这就是所谓的**帕累托法则**（Pareto principle）的一个实例。该法则援引自意大利经济学家、社会学家维夫雷多·帕累托（Vilfredo Pareto，1848—1923），他发现意大利的一小部分人口控制了意大利的大部分财富。在软件工程领域，帕累托法则认为，通过对一个集中区域施加作用，往往就可以明显地改变结果。

软件测试的另一种方法称为**基本路径测试**（basis path testing），这种方法要开发出一组测试数据，并且这组数据要能保证软件中的每条指令都能至少执行一次。人们已经用数学领域的图论开发出了确定这种测试数据集的技术。因此，虽然不可能保证通过软件系统的每条路径都得到测试，但是可以做到在测试过程中，系统的每条语句都至少能执行一次。

基于帕累托法则和基本路径测试的技术都依赖于对被测试软件的内部构成的理解，因此这类测试都属于所谓的**白盒测试**（glass-box testing）这一类，这也就意味着软件测试人员要了解软件的内部结构，在设计测试的时候要利用到这些知识。相反，还有一类测试称为**黑盒测试**（black-box testing），这类测试并不依赖于对软件内部构成的了解。简而言之，黑盒测试是从用户的角度来完成的。在黑盒测试过程中，测试人员并不关心软件本身是如何工作的，而只注重软件在精确度和时间性方面是否能正确执行。

黑盒测试的一种方法是称为**边界值分析**（boundary value analysis）的技术，它包括确定数据范围（即**等价类**，软件应以类似的方式操作它们）和用这些范围的边界数据测试软件。例如，如果软件需要接受指定范围内的输入值，那么就可以用这个范围内的最低值和最高值对该软件进行测试；或者如果软件需要协调多个活动，那么就应针对尽可能大的活动的集合进行测试。基于的理论是：通过确定等价类，因为对于一个等价类内的几个例子的正确操作往往可验证整个类的软件，所以测试用例的数量可以最小化。此外，确定一个类内错误的最佳机会是使用类边界上的数据。

黑盒测试的另外一种方法是**β测试**（beta testing），在得到产品最终版本并向市场发布之前，软件的初步版本被发给特定用户，以了解软件在现实环境中的表现如何。注意，在开发者地点进行的类似测试称为**α测试**（alpha testing）。β测试的优点远远不止传统的排查错误。通过这种测试所获得的普通用户的反馈意见（无论正面或负面）将有助于调整市场策略。此外，早些时候发布的beta版的软件有助于其他软件开发者设计出与之兼容的产品。例如，就PC市场的新操作系统来说，其beta版本的发布会鼓励与之兼容的工具软件的开发，所以最终版的操作系统上市时，就已经有与之相配的软件产品出现。此外，beta版软件的存在会在市场上造成一种对软件产品的期待（一种增加推广和销量的氛围）。

问题与练习

1. 软件开发组织内的SQA小组的作用是什么？
2. 人性是以什么方式与质量保证对立的？
3. 说出两种在开发过程中用来加强质量的主题。
4. 在测试软件时，一个成功的测试是指发现了错误，还是没有发现错误的测试？
5. 为了确定系统中的哪些模块应该接受比其他模块更为彻底的测试，你会建议采用什么技术？
6. 一个软件包设计用来对不超过100项的表进行排序，请问，对此软件包采用什么样的测试最为合适？

7.7 文档编制

如果人们不能学会使用和维护软件系统，那么这个软件系统也就没多大的用处。因此，文档是软件包的一个重要的部分，而文档的编写也就成了软件工程领域里的一个重要课题。

软件文档有 3 个用途，因而也就可以将文档划分为 3 类：用户文档、系统文档以及技术文档。**用户文档**（user documentation）用来解释软件的特性，并描述如何使用软件。所设计的用户文档是给用户（所以称为用户文档）浏览的，因而其编写方式上采用的是应用方面的术语。

今天，用户文档被公认为是一种重要的市场工具。好的用户文档加上精心设计的用户界面，使得软件更容易为人们所接受，这样就提高了销售量。正因为认识到这一点，许多软件开发人员聘请熟悉技术的编写人员为其编制产品的文档，或者将自己软件产品的初级版本提供给独立作者。这样一来，当开始向公众发布软件正式版时，书店里也同时有了关于如何使用该软件的书籍。

用户文档传统上是纸质书籍或小册子形式，但是许多情况下同样的信息也包含在该软件中成为其组成部分。这使读者在使用软件时能参考文档。此时，信息可能分割成小单元，有时称为帮助包。当用户在多个命令之间犹豫不决时，帮助包中的信息可以自动出现在屏幕上。

系统文档（system documentation）用来描述软件的内部构成，便于在软件日后的生命周期中进行维护。系统文档的一个主要部分是系统中所有程序的源代码。这些源程序应以易读的格

式呈现，这一点很重要。这就是软件工程师支持采用精心设计的高级编程语言，采用注释语句对程序进行注释，采用以逻辑单元表示模块的模块化设计的原因。实际上，大多数软件开发公司都有一定的要求其员工在编写程序的时候遵循的约定。例如，使程序编写得有条理的缩进约定，确立变量、常量、对象以及类等不同程序结构的命名约定，以及保证所有程序都能有效地文档化的文档编写约定。这些约定在整个公司的软件中都是统一的，这样最终就能简化软件的维护过程。

系统文档的另外一个组成部分是设计文档的记录，其中包括了软件需求规格说明文档和显示这些规格说明在设计期间被如何获得的记录。这些信息对于软件的维护是有帮助的，因为它指明了软件为何要这样实现，同时这些信息也降低了这样一种可能性，即在维护阶段所作出的变更会破坏系统的完整性。

技术文档（technical documentation）用来描述软件系统应如何安装以及相关的服务（如调整操作参数、安装更新以及将出现的问题反馈给软件开发人员等）。软件的技术文档与汽车工业中提供给汽车修理工的文档类似。这种文档不讨论汽车是怎样设计和构造的（这类似于软件的系统文档），也不解释如何驾驶汽车和操作汽车加热/制冷系统（这类似于软件的用户文档），而是用来描述如何维护汽车的配件，例如如何替换变速箱，或者如何解决断断续续的电气方面的问题。

在PC机领域里，软件的技术文档和用户文档之间的差异就变得比较模糊了，这是因为用户通常自己安装和维护软件。然而，在多用户的环境中，这种差异就更明显了，因为这种情况下，技术文档是提供给系统管理员使用的。系统管理员在其权限下负责所有软件的维护，允许用户将软件包作为抽象工具访问。

问题与练习

1. 软件可以以哪些形式文档化？
2. 系统文档应在软件生命周期的哪个（哪些）阶段进行准备？
3. 程序和它的文档相比，哪个更重要？

7.8　人机界面

回顾一下7.2节，其中讲到需求分析阶段的一项任务是定义要开发的软件系统将如何与它的环境进行交互。本节我们将考虑与这个交互相关的主题，那就是当它涉及与人交流时的情况，这是一个意义深远的主题。毕竟，应该允许用户把软件系统当做一个抽象工具来使用。这个工具应该易于使用，最小化（理想情况下消灭）用户与系统间的交流错误。这意味着系统界面的设计应方便用户的使用，而不仅是作为软件系统的权宜之计。

良好的界面设计非常重要，因为与系统的其他特性相比，系统界面更容易给用户留下深刻的印象。毕竟，用户往往会从系统的可用性角度来审视一个系统，而不是从它如何巧妙地执行其内部任务这个角度。从用户的视角来说，他们可能会根据系统界面在具有竞争性的系统之间作出选择。因此，系统界面的设计可能成为判定一个软件工程项目是否成功的最终决定因素。

由于这些原因，人机界面在软件开发项目的需求分析阶段已经成为一个很重要的关注点，并是软件工程的一个不断成长的子领域。事实上，有些人认为人机界面的研究是一个完全独立的领域。

智能手机界面便得益于这一领域的研究。为了能够实现口袋大小的袖珍设备，传统人机界

面的元素（标准尺寸的键盘、鼠标、滚动条、菜单）已被新的方式所取代。（新的方式包括在触摸屏上操作的手势、语音命令、具有高级的单词和短语自动输入功能的虚拟键盘。）尽管这标志了重大的进展，但大多数智能手机用户认为进一步创新的空间还很大

对人机界面设计的研究主要来自于称为**人体工程学**（ergonomic）和**知行学**（cognetic）的工程领域，人体工程学处理协调人类体能的设计系统，知行学处理协调人类精神能力的设计系统。这两个学科中，人体工程学更好理解一些，主要是因为人类已经跟机器打了几个世纪的交道。这些例子有：古代工具、武器和运输系统。这些历史大部分是不证自明的，但是有时人体工程学的应用与直觉是相反的。一个经常被提到的例子就是打字机键盘（现在已经衍生为电脑键盘）的设计，其中键被有意排列，以降低打字员的速度，这样早期机器上使用的分层机械系统就不会卡住。

相反，与机器的精神交互是一个相对较新的现象，因此知行学在富有成效的研究和洞察力启发方面拥有更高的潜力。通常这些研究成果更具有精妙之处。比如，从表面上看人类的良好习惯有助于提高效率，但有些习惯也会导致一些错误，即使界面设计本意上是要解决问题的。考虑一下用户要求操作系统删除一个文件的过程，为了防止误删，大部分界面都会要求用户确认一个请求，这可能会通过一个类似“你是否真的想删除这个文件”的信息要求确认。乍一看，这个确认信息好像解决了误删的问题，但是使用了这个系统一段时间后，用户会养成习惯，自动回答“是”。这样，这个删除文件的任务就从包含删除命令和对问题思考后的响应的两步过程，变成了“删除—是”的一步处理过程，这就意味着当用户意识到提交了错误的删除要求时，这个请求其实已经被确认，文件也已经被删除。

当人们需要使用几个应用软件包时，习惯的形成也可能会带来问题。这些软件包的界面可能相似，但还是有些不同的。相似的用户操作可能会导致不同的系统响应，或类似的系统响应可能需要不同的用户操作。在这种情况下，在某种应用软件上养成的操作习惯可能会在其他应用软件上导致错误的发生。

另外一个与人机界面设计研究有关的人类特质就是人类注意力的狭隘性，也就是当集中度增加时，人类注意力往往变得更加专注。随着人类越来越专注于手头上的工作，打破这种专注也越来越困难。1972年，一架商务飞机因为飞行员太过专注于降落器的问题（实际上，是在改变降落齿轮指示灯的过程中），尽管当时在驾驶舱里的警报已经响了，飞机还是笔直地撞向地面，造成了空难。

个人计算机的界面中经常会出现一些小状况。比如，大多数键盘提供大小写灯，这是为了显示键盘处在大写键锁定模式下（即“大写锁定”键被按了）。但是，如果有人不小心按了大小写按键，直到奇异的字符出现在屏幕上，用户才会注意到灯的变化。即使如此，用户依然会迷茫一会儿才会发现问题的原因。从某种意义上来说，用户看不到大小写灯的变化是很正常的，因为键盘的指示灯不在用户的视线范围之内。但是，通常用户也不能注意到直接放置在他们视线中的指示灯。比如，用户会专注于他们的工作而无法发现显示器上光标的形状变化，即使观察光标是他们的工作之一。

还有另外一个在界面设计阶段必须预先考虑的人类特质，即并行处理多个事情时有限的思考能力。在1956年《心理评论》的一篇文章中，George A. Miller的研究表明，人类大脑在同一时间最多处理7个细节问题。因此，界面被设计成：当决定需要时，界面上要呈现所有相关的信息，而不应依赖人类用户的记忆，这是非常重要的。特别地，若要求人类记住先前屏幕图像中的精确细节，这就是很糟糕的设计。更进一步地，如果界面需要用户在屏幕图像间进行大量导航，用户会变得很迷惑。因此，屏幕图像的内容和安排成为了一个重要的设计问题。

尽管人体工程学和知行学的应用使得人机界面设计折射出独特的韵味，但这个领域还是包含着软件工程中很多更加传统的主题。特别地，搜索度量在界面设计领域和更传统的软件工程领域中具有同样的重要性。界面可以度量的特性包括了解一个界面所需的时间、在界面上完成任务所需的时间、用户界面出错的概率、一段时间不用后用户使用界面的熟练程度，甚至是一些诸如用户对界面喜好程度的主观特性。

GOMS模型最初在1954年提出，它是人机界面设计领域里度量搜索的范例。这个模型的基础方法论是依据用户的目标（如删除文档中的某个字）、操作（如单击鼠标按键）、方法（如双击鼠标，然后按删除键）和选择规则（实现相同目标的两种方法间的选择）分析任务。实际上这就是GOMS缩写的起源——goal（目标）、operator（操作）、method（方法）以及selection rule（选择规则）。简言之，GOMS就是一种把用户使用一个界面的动作分析成基本步骤序列（按键、移动鼠标和作出决定）的方法论。每个基本步骤的性能都被赋予一个精确的时间段，这样通过把任务中赋予每个步骤的时间相加，从完成相似任务每个界面所需的时间这个角度来看，GMOS提供了一种比较不同的提议界面的方法。

理解类似于GMOS的系统的技术细节不是我们当前的研究目的，我们示例的要点是，GOMS以人类行为（移动手、作出决定等）特性为基础。事实上，GMOS的发展起初被认为是心理学中的主题。这样GMOS重新强调了人类特性在人机界面设计领域中，以及在那些从传统软件工程延伸的主题中所起的作用。

在可预见的未来，人机界面设计肯定是一个活跃的研究领域。处理当今GUI的许多问题依然没有得到解决，大量附加问题潜存于三维界面的使用中（这样的3D界面已经出现）。实际上，因为这些界面承诺结合语音和与三维视觉的触摸交流，所以潜在问题的范围是巨大的。

问题与练习

1. a. 说出人机界面设计领域中的人体工程学的应用。
   b. 说出人机界面设计领域中的知行学的应用。
2. 对比智能手机与台式机的人机界面，一个显著区别便是滚动显示区域所运用的技术。在台式机上，滚动通常通过用鼠标拖曳显示区域下方或右侧的滚动条来实现，而智能手机通常不使用滚动条。（就算使用，它们通常也显示为细线，以表明当前哪一部分可见。）因此，智能手机界面上的滚动通过在显示屏上进行的滑动触摸手势实现。
   a. 根据人体工程学，可以提出什么论据来支持这一区别？
   b. 根据知行学，可以提出什么论据来支持这一区别？
3. 人机界面设计与更传统的软件工程领域有什么不同？
4. 说出在设计人机界面时要考虑的人类的3个特征。

7.9　软件所有权和责任

大多数人都会同意这样一个观点，即公司或个人投资开发高质量的软件，都可以从中获利，得到回报，否则就很可能没有人愿意从事开发社会所需的软件的工作了。简言之，软件开发者需要对他们生产的软件拥有一定的所有权。

提供这种所有权的法律措施归类于**知识产权**法，其中许多依据的都是完善确立的版权法和专利法原则。实际上，版权和专利的目的是允许“产品”的开发者在向公众发布产品时，其所有权得到保护。因此，某一产品的开发人员（无论是个人还是公司）应在其创作的所有作品中

加入版权声明，并以此来声明其对该产品的所有权。（这里的作品包括需求规范、设计文档、源代码、测试计划和最终产品。注意，版权声明要放在最终产品的某个显著位置。）版权声明清晰地确定了物主身份、被授权使用该产品的人员及其他限制。此外，开发人员的权利通过**软件许可**（software license）中的法律条款正式表述。

软件许可是软件所有者与软件用户之间的一份法律协议，它为用户使用这一产品授予一定的使用许可，但不允许转让知识产权。这些协议非常详细地解释了各方的权利和义务。因此，在安装和使用某一软件产品前，仔细阅读和理解软件许可中的条款是非常重要的。

尽管版权和软件许可协议为禁止直接复制和非授权使用软件提供了法律保护，但它们通常不足以禁止另一方独立开发与该产品功能几近相同的软件产品。令人遗憾的是，多年来很多真正具有创新性的软件产品的开发人员却无法从他自己的发明中充分获利（其中两个著名的例子便是电子制表软件和Web浏览器）。在大多数情况下，往往是另一家公司成功地开发了具有竞争力的产品，并占据了具有绝对优势的市场份额。就这一方面来说，阻止竞争对手侵扰的一个法律依据就是专利法。

专利法的建立是为允许发明者从他的发明中获得商业上的利益。为了获得专利，发明者必须透露发明的细节，并说明这是新的、有用的，并且对于类似背景下的其他人不是轻而易举做到的（对于软件来说，这一需求非常具有挑战性）。如果一个专利被授权，那么在一段有限的时期内发明者就被赋予了这样的权力，即防止其他人制造、使用、销售或引入专利。这段时间一般是专利申请被提出之日起的20年。

采用专利的一个问题是，获取专利是一个昂贵的、费时的过程，通常历时几年。在这段时间内，软件产品可能已经被淘汰了，而直到专利被授权，申请者手中只有靠不住的权限去阻止别人盗用其产品。

在软件工程过程中，明确版权、软件许可和专利是极其重要的。在开发某一软件产品时，软件工程师通常会集成取自其他产品的软件，它可能是一个完整的产品、一部分组件，或者是通过因特网下载的一部分源代码。然而，如果在此过程中未能尊重知识产权，则将可能导致巨大的损失和严重的后果。例如，2004年一个不太出名的公司NPT成功地赢得了一场法律诉讼，它控告RIM（Research In Motion，黑莓智能手机制造商）在邮件系统中侵犯了它的一些关键技术的专利权。这一诉讼的判决结果包括暂停RIM向美国所有黑莓用户提供电子邮件服务的禁令！最终，RIM与NPT达成协议并支付给NPT总共6.125亿美元，从而避免了关停的命运。

最后，我们应当提出责任的问题。为了使自己免于责任，软件开发者通常会在其产品上附带免责声明，用以说明其责任的限制。诸如"因使用本软件所造成的任何损失，本公司概不负责"这样的声明比较常见。然而，如果控方能够举出被告的疏忽之处，法庭很少会认可这类声明。所以，责任案件容易集中在被告是否对生产的产品给予了相应的关照程度。一个在开发字处理系统的情况下认为可以接受的关照程度，如果放在核反应堆的控制软件的开发上，就可能被认为是一种疏忽。因此，对软件责任声明的最好防护之一就是，在软件的开发过程中，运用合理的软件工程准则，采用与软件应用相适应的关注程度，产生并维护验证这些努力的记录。

问题与练习

1. 在需求规范、设计文档、源代码和最终产品中，版权声明的意义是什么？
2. 版权法、专利法，这些法规的制定对社会有何益处？
3. 免责声明中的什么内容将不会被法庭认可？

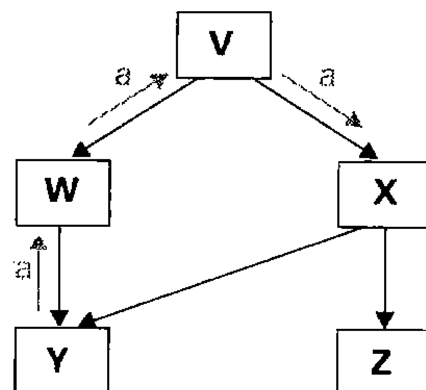
复习题

（带*的题目涉及选读小节的内容。）

1. 举出一个例子，说明软件开发时所做的努力是如何在日后的软件维护中得到回报的。
2. 什么是演化式原型开发？
3. 试解释缺少度量某些软件特性的度量学是如何影响软件工程学科的。
4. 你是否认为度量软件系统复杂性的度量标准是积累的？积累的意思是：整个系统的复杂性是其各部分的复杂性之和，解释你的答案。
5. 你是否认为度量软件系统复杂性的度量标准是可交换的？可交换的意思是：如果系统最初开发了X特性，后来加入了Y特性，或者是如果原先开发了Y特性，后来增加了X特性，那么整个系统的复杂性是相同的，解释你的答案。
6. 软件工程是如何区别于诸如电子、机械工程之类传统工程领域的？
7. a. 给出软件开发中采用传统瀑布模型的缺点。
   b. 给出软件开发中采用传统瀑布模型的优点。
8. 开源开发是一个自顶向下或者自底向上的方法学吗？请解释你的答案。
9. 试描述常量的使用是如何比字面量的使用更能简化软件维护的？
10. 耦合和内聚的区别是什么？哪个应该最小化？哪个应该最大化？为什么？
11. 从日常生活中选取一个对象，依据功能内聚和逻辑内聚来分析其组成部分。
12. 试将由一条简单的goto语句所造成的两个程序段间的耦合与由过程调用所引起的耦合进行比较。
13. 在第6章中我们已经知道，参数可以通过按值传递或按引用传递这两种方式传递给过程。哪一种提供了更为复杂的数据耦合形式？请解释你的答案。
14. 如果一个大型的软件系统中的数据元素都设计成全局数据，那么在维护阶段中可能会出现什么问题？
15. 在面向对象程序中，声明一个实例变量是公有的或是私有的对数据耦合意味着什么？对于声明实例变量为私有的这一偏好，其背后的基本原理是什么？

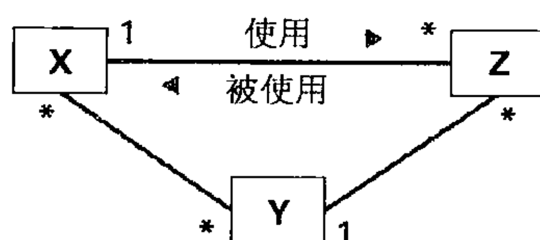
*16 举出一个涉及并行处理环境下发生的数据耦合的问题。

17. 按如下结构图回答下列问题：

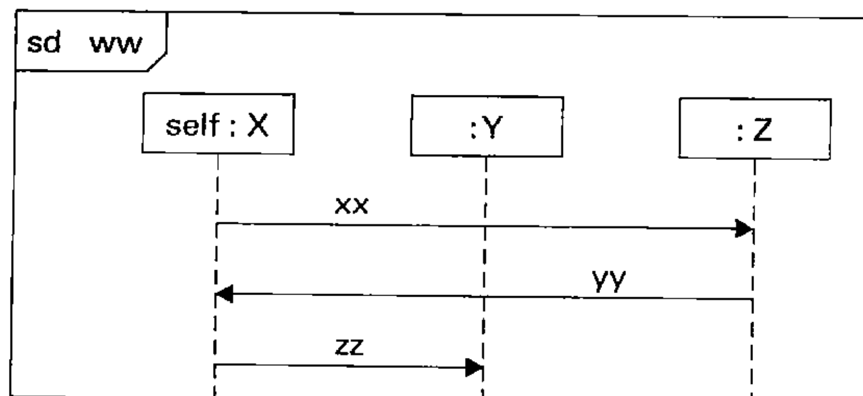


    a. 模块Y把控制返回到哪个模块？
    b. 模块Z把控制返回到哪个模块？
    c. 模块W和模块X是通过控制耦合链接起来的吗？
    d. 模块W和模块X是通过数据耦合链接起来的吗？
    e. 哪些数据是由模块W和模块Y共享的？
    f. 模块W和模块Z以什么方式相关联？
18. 用一个结构图来表示为小商店（也许是在人流量较大的社区开的一家私人古董店）开发的一个简单库存/账务系统的过程结构。请问，出于营业税的变动，你必须要修改系统中的哪些模块？如果想给以前的顾客邮寄广告，那么当你决定要维护一个老顾客的记录时，应该对哪些模块进行修改？
19. 对上题设计一个面向对象的解决办法，并用一个类图进行表示。
20. 画出一个简单的类图，表示杂志出版商、杂志和订阅者之间的关系。其实，只要在表示每个类的相应矩形框中描述类名即可。
21. 什么是UML？UML的作用是什么？请详细解释与“M”这一字母对应的词。
22. 画出一个简单的用例图，描述图书馆的顾客使用图书馆的方式。
23. 请画出一个序列图，表示当公用事业机构给客户发送账单时，继而发生的交互序列。
24. 画出一个简单的数据流图，用来描述当一个交易完成时，自动库存系统里所出现的数据流。
25. 请将类图所表示的信息与序列图所表示的信息进行比较。
26. 请说明一对多联系与多对多联系有何不同？

27. 请举出一个本章中没有提到的一对多联系的例子。举出一个本章中没有提到的多对多联系的例子。
28. 基于图7-10中的信息，想象一下在看望病人的过程中医生和病人间可能发生的交互序列。画出表示这个序列的序列图。
29. 画出一个类图，表示饭店里服务生和顾客之间的关系。
30. 请画出一个类图，用来表示杂志、杂志出版商和订阅者之间的关系。并提供每个类的实例变量和方法。
31. 扩展图7-5中的序列图，显示这样的序列：PlayerA成功地返回了PlayerB的球，但PlayerB未能成功返回这个球。
32. 基于如下类图回答下列问题，类图表示的是工具、工具的用户以及工具的生产厂商间的关联。



   a. 哪个类（X、Y和Z）表示工具、用户和厂商？验证你的答案。
   b. 工具能被多于1个用户使用吗？
   c. 工具能被多于1个厂商制造吗？
   d. 是否是每个用户使用由多个厂商制造的工具？
33. 根据下面的各种情况，判断所述的活动是与序列图、用例图有关，还是与类图有关。
   a. 确定用户将与系统交互的方式。
   b. 确定系统中类之间的关系。
   c. 确定完成某一任务时对象的交互方式。
34. 基于下面的序列图，回答下列问题。



   a. 什么类含有名为ww的方法？
   b. 什么类含有名为xx的方法？
   c. 在序列中，“类型”为Z的对象会与“类型”为Y的对象直接通信吗？
35. 请画出一个序列图，说明对象A调用对象B中的方法bb，B执行请求的动作并返回控制给A，然后A再调用对象B中的方法cc。
36. 扩展对前面问题的解决方法，表明只有当变量“continue”为真时，A才能调用方法bb，在B返回控制后，只要“continue”继续为真，A就可以继续调用bb。
37. 请画出一个类图，用来描述这样一个事实，即Truck（卡车）类和Automobile（小汽车）类都是Vehicle（汽车）类的泛化。
38. 基于图7-12，什么额外实例变量会包含在“类型”为SurgicalRecord的对象中？对于类型为OfficeVisitRecord呢？
39. 为什么继承并不总是实现类泛化的最佳方式？
40. 请举出软件工程领域以外的一些设计模式。
41. 总结设计模式在软件工程中的作用。
42. 在什么程度上来说，一个典型的高级程序设计语言中的控制结构（如if-then-else、while等）就是一个小型的设计模式？
43. 以下情况中，哪个涉及了帕累托法则？并解释你的答案。
   a. 一粒老鼠屎搞坏一锅粥。
   b. 每个电台集中于一种特定的形式，如摇滚乐、古典音乐、谈话节目等。
   c. 在选举活动中，候选人非常明智地将其重点放在之前投他们票的那部分选民上。
44. 软件工程师希望大型软件系统在错误的内容上是同种类型的，还是不同类型的？请解释你的答案。
45. 黑盒测试与白盒测试的区别是什么？
46. 试举出一些在软件工程以外的领域中发生的与黑盒测试和白盒测试类似的事件。
47. 开放源码开发与beta测试有何区别？（考虑白盒测试和黑盒测试。）
48. 假定在一个大型系统快要完成最后的测试前，故意放入100个错误。此外，还假定在最后的测试期间发现并纠正了200个错误，而其中的50个错误属于故意放入系统中的。请问，如果接下来那些剩下的50个已知错误也被纠正了，那么你估计系统中还有多少个没有发现的错误？为什么？
49. 什么是GOMS？
50. 什么是人体工程学？什么是知行学？

51. 对比智能手机与台式机的人机交互界面，一个区别就在于缩放显示屏上图像的技术。在台式机上，缩放通常通过拖动一个独立于显示区的滑块来实现，或者通过菜单或工具栏选项实现。在智能手机上，缩放是这样实现的：用大拇指和食指同时触摸显示屏，然后改变两个触摸点间的距离。
    a. 根据人体工程学，可以提出什么论据来支持这一区别？
    b. 根据知行学，可以提出什么论据来支持这一区别？
52. 在什么情况下，传统的版权法无法保护软件开发者的投资？
53. 在什么情况下，软件开发者不能成功获得专利？

社会问题

下面的问题有助于分析一些与计算领域相关的伦理、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. a. 分析员玛丽被分配了一个任务，即要实现一个系统。通过该系统可以将医疗档案存放在联网的计算机上。依据她的观点来看，系统安全性方面的设计存在着缺陷，但是由于公司财力方面的原因，她所提出的想法被否决了，而且公司要求使用她认为不太合适的安全系统来继续该项目。这种情况下，她该怎么办？为什么？
   b. 假设分析员玛丽按照吩咐实现了该系统，而现在她发现有非授权人员在检索医疗档案。这时她该怎么办？对于这样一种侵犯安全的情况，她将负多大责任？
   c. 假设分析员玛丽没有听从老板的安排而拒绝再开发这个系统，并且义无反顾地将设计缺陷公布于众，结果导致公司的财务紧张，许多无辜的员工失去工作。分析员玛丽的行为对吗？如果玛丽仅仅是整个设计组的一名成员，她并不了解公司正在花费大的精力在别的地方开发一套有效的安全系统，而这套系统将会用在玛丽正在开发的系统上。那么又会怎么样？这种情况是如何改变你对玛丽行为的判断的？（需要注意的是，玛丽对这种情况的观点和以前一样。）
2. 当一个大型软件系统由许多人一起开发时，如何分配责任？是否有一种层次型的责任？是否有不同程度的责任？
3. 我们知道，大型的复杂软件系统通常是许多成员一起开发的，其中很少有人能够对整体系统有一个全面的了解。那么对于一名员工来说，他对系统的功能没有完全了解，却要为该项目出力，这么做在道德上是否合适？
4. 某人对其成果最终为他人所用，应当负多大的责任？
5. 在计算机专业人员与客户之间的关系中，专业人员的责任是实现客户的需求，还是对客户的需求加以指导？如果专业人员预见到客户的要求会导致缺乏职业道德的结果发生，该怎么办？例如，客户可能为了提高效率希望走捷径，而专业人员预见到如果采用走捷径的方式，可能会成为产生数据错误或系统误用的根源。如果客户坚持这么做，那么专业人员是否就没有责任？
6. 如果技术的发展太过迅猛，发明者还未来得及从他的发明中获利，新的发明却已紧随而来，取而代之。这样将会发生什么？这种获利对激励发明者而言是必需的吗？开放源码开发的成功与你的答案有什么关系？免费的高质量软件足以持续支撑现实的需求吗？
7. 计算机革命能否有助于（或者说帮助解决）世界能源问题？对其他的一些大规模问题，如饥饿和贫穷等，情况又会如何？
8. 技术是否会无限期地发展下去？是否有什么因素会逆转社会对技术的这种依赖？如果社会继续推进技术无限期地发展下去，那么结果将会怎样？

9. 如果你有一台时间机器，你想要生活在哪个历史时间段中？是否有你想带走的当前技术？一种技术能与另一种技术分开吗？一边反对全球变暖，一边接受现代医学治疗，这现实吗？
10. 智能手机上的许多应用会自动集成其他应用提供的服务。这种集成可能会将进入一个应用的信息与另一个应用共享。这种集成的好处是什么？过多地集成会导致什么问题吗？

课外阅读

Alexander, C., S. Ishikawa, and M. Silverstein. *A Pattern Language*. New York: Oxford University Press, 1977.

Beck, K. *Extreme Programming Explained: Embrace Change*, 2nd ed. Boston, MA: Addison-Wesley, 2004.

Bowman, D. A., E. Kruijff, J. J. Laviola, Jr., and I. Poupyrev. *3D User Interfaces Theory and Practice*. Boston, MA: Addison-wesley, 2005.

Braude, E. *Software Design: From Programming to Architecture*. NewYork: Wiley, 2004.

Bruegge, B. and A. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java*, 3rd ed. Boston, MA: Addison-Wesley, 2010.

Cockburn, A. *Agile Software Development: The Cooperative Game*, 2nd ed. Boston, MA: Addison-Wesley, 2006.

Fox, C. *Introduction to Software Engineering Design: Processes, Principles and Patterns with UML2*. Boston, MA: Addison-Wesley, 2007.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1995.

Maurer, P. M. *Component-Level Programming*. Upper Saddle River, NJ: Prentice-Hall, 2003.

Pfleeger, S. L. and J. M. Atlee. *Software Engineering: Theory and Practice*, 4th ed. Upper Saddle River, NJ: Prentice-Hall, 2010.

Pilone, D. *UML 2.0 in a Nutshell*. Cambridge, MA: O'Reilly Media, 2005.

Pressman, R. S. *Software Engineering: A Practitioner's Approach*, 7th ed. New York: McGraw-Hill, 2009.

Schach, S. R. *Classical and Object-Oriented Software Engineering*, 8th ed. New York: McGraw-Hill, 2010.

Shalloway, A. and J. R. Trott. *Design Patterns Explained*, 2nd ed. Boston, MA: Addison-Wesley, 2005.

Shneiderman, B., C. Plaisant, Cohen, M, and Jacobs S. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 5th ed. Boston, MA: Addison-Wesley, 2009.

Sommerville, I. *Software Engineering*, 8th ed. Boston, MA: Addison-Wesley, 2006.

第8章

数据抽象

本章将要研究的是如何对数据组织形式进行模拟，这门学科称为数据结构，它不同于由计算机内存所提供的以一个个单元来组织数据的方式。其目标是让数据的使用者将数据集视为一种抽象的工具来访问，而不是从计算机内存中的数据组织的角度去考虑问题。这方面的研究工作将向我们展示，构造这种抽象工具的需求是如何产生对象和面向对象编程概念的。

本章内容

第6章已经介绍了数据结构这个概念。在那一章中，我们已经了解到：程序员可以利用高级程序设计语言所提供的技术来表示算法，就好像所操作数据的存储方式并不是按照一个个单元在内存中存放。我们还学习到，编程语言所支持的数据结构称为基本结构。本章将讨论一种技术，利用这种技术能够构建和操作与语言的基本结构不同的数据结构，该研究能够使我们从传统的数据结构过渡到面向对象的范型。贯穿这项工作进展的潜在主题是抽象工具的构建。

8.1 数据结构基础

这里，先介绍一些基本的数据结构并作为后续几节的例子。

8.1.1 数组

在6.2节中，已经介绍了同构数组和异构数组这两个数据结构。**同构数组**（homogeneous array）是一种“矩形的”数据块，其项具有相同的类型。具体来说，一个二维同构数组由行与列组成，其中，项的位置由一对下标确定，即第一个下标值确定项的行位置，第二个下标值确定项的列位置。例如，用一个矩形数组表示销售人员的每月销售额，每行的项代表的是某个销售人员每月的销售额，每列的项代表的某个月每个销售人员的销售额。这样一来，第3行第1列的项就可以表示第3个销售人员第1个月的销售额。

与同构数组不同，**异构数组**（heterogeneous array）是一个可能具有不同类型的项块。块里的项通常称为**部件**（component）。例如，用一个异构数组的数据块表示一个员工，其部件可能有三项：员工的名字（字符型）、年龄（整型）以及技能等级（实型）。

8.1.2 列表、栈和队列

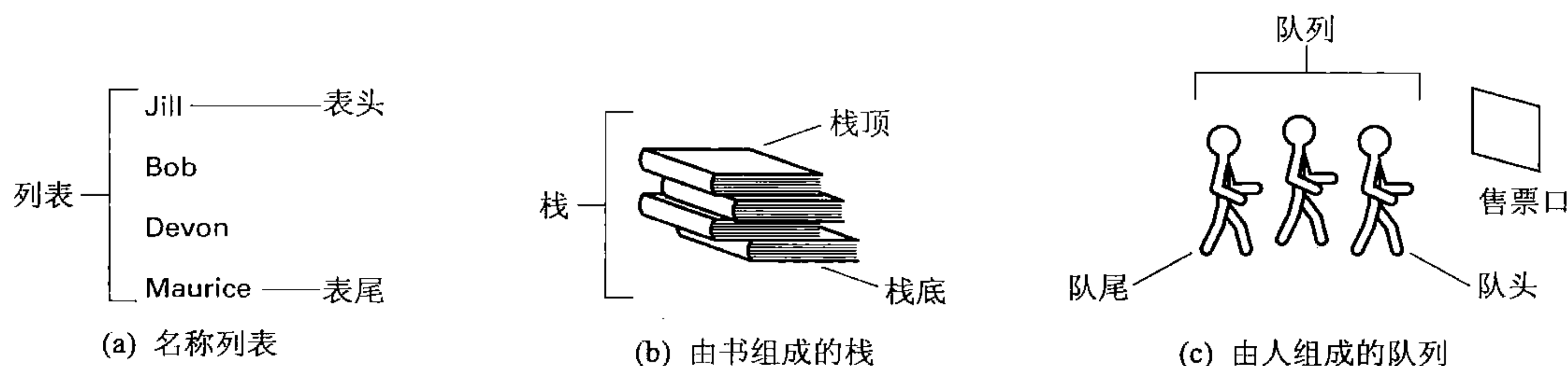
列表（list）是这样的一组数据，其表项按顺序排列（如图8-1a所示）。一个表的开头称为**表头**（head），表的尾端称为**表尾**（tail）。

几乎所有的数据集合都可以看成列表。例如，文本可以被看成符号的列表，二维数组可以看成是行的列表，CD上记录的音乐可以看成是声音的列表。更为常见的例子包括客人清单、购物清单、班级注册表、存货表等。与列表相关的操作视情况而定。在某些情况下，我们可能需要从列表中移除项，向列表中增加项，每次“处理”列表中的一个项，改变项在列表中的排列，或者查找某个特殊的数据项是否在列表中。我们将在本章的后面讨论这些操作。

通过严格限制列表中项的访问方式，我们可以得到两种特殊类型的表，称为栈和队列。**栈**（stack）是这样的一种列表，该表的项只能在表头进行添加和删除（如图8-1b所示）。用通俗的术语来表示，栈的头称为**栈顶**（top），栈的尾称为**栈底**（bottom或base）。在栈顶增加一个新的项称为**入栈**（pushing），在栈顶删除一个项称为**出栈**（popping）。注意，最后入栈的数据最先出栈，这样就可以得到：栈是LIFO（Last-In，First-Out，后进先出，读作“LIE-foe”）的结构。

对于那些检索次序与存储次序相反的存储数据项而言，这种后进先出特性意味着栈是理想的选择，所以栈经常被用作回溯活动的基础。[术语**回溯**（backtracking）是指退出系统的过程，它与进入系统的次序相反。一个经典的例子是：为了找到走出森林的路径而原路返回。] 例如，思考一下支持递归过程所需的基本结构，在每一个新活动开始时，先前的活动必须保存下来。而且，在每一个活动结束时，必须检索前一个保存的活动。这样，如果当活动被保存时就会压入栈中，那么每次需要检索一个活动时，合适的活动将处在栈顶。

队列（queue）是这样的一种列表，其表项只能从表头删除，新表项只能从表尾插入。这种数据结构的例子有，戏院门口排队等待购票的一队人（见图8-1c），这里，位于队列头的人先购票，而新到的人必须到队尾进行排队购票。在第3章中，我们已经遇到过这种数据结构，在那节中，批处理系统所存放的作业必须在所谓的作业队列中进行排队，等待执行。可以得出这样的结论：与栈不同，先进队列的项会先从队列中删除，就是说队列是FIFO（First-In，First-Out，先进先出，读作“FIE-foe”）的结构，这意味着表项以它们存储的顺序从队列中删除。



(a) 名称列表　(b) 由书组成的栈　(c) 由人组成的队列

图8-1 表、栈和队列

正如第1章中介绍的，队列常被用作缓冲区的基本结构，缓冲区是从一处传送到另一处的数据临时放置的存储区域。当一项数据到达了缓冲区，它就被放置在队列的末尾。当需要转发数据项到达最终的目的地时，它们按其在队列头部出现的次序被转发。因此，数据转发的次序就是它们到达的次序。

8.1.3 树

树（tree）是这样的一个数据集合，其项具有层次化的组织形式，很像一个典型的公司组织

关系图（如图8-2所示）。这种组织图中，顶部表示总裁，由分支线下连到副总裁，副总裁又连到地区经理，等等。对树的这种直观性的定义，我们还要加上一个限制性条件，即（参照组织图）公司的任何一个员工只有一个直接上司。也就是说，组织中的不同分支不会在下一层相遇。（第6章已经举过几个树的例子，是以语法分析树的形式介绍的。）

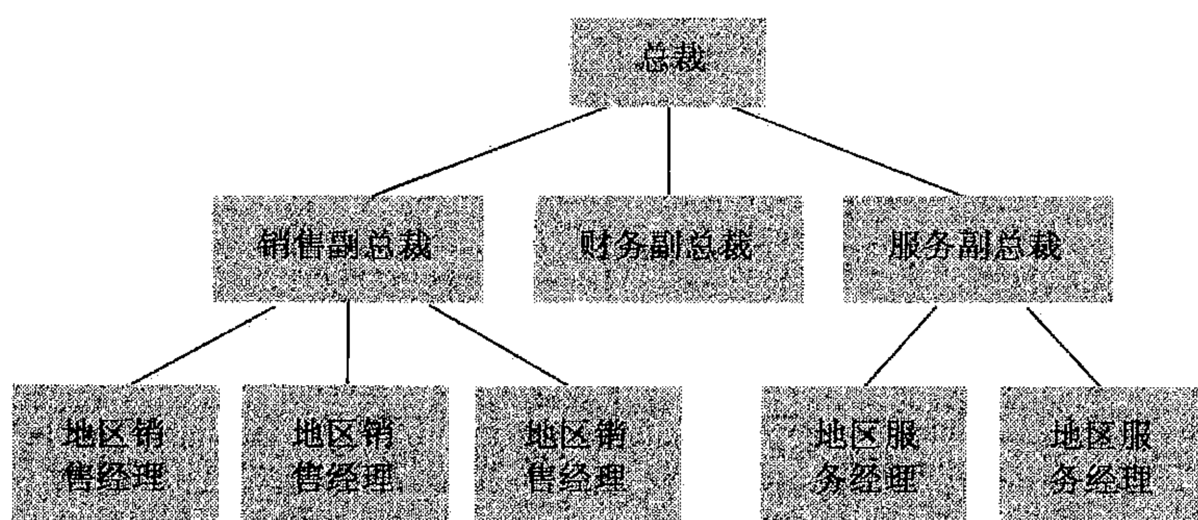


图8-2 组织图的一个例子

树中的每一个位置称为一个**节点**(node)(如图8-3所示)。树顶部的那个节点称为**根节点**(root node)——如果我们把图倒过来看，这个节点就表示了树的根。另一端点处的节点称为**终端节点**（terminal node），有时也称为**叶子节点**（leaf node）。我们常将从根到叶子的最长路径上的节点数称为树的**深度**（depth）。换句话说，一个树的深度就是该树所包含的层数。

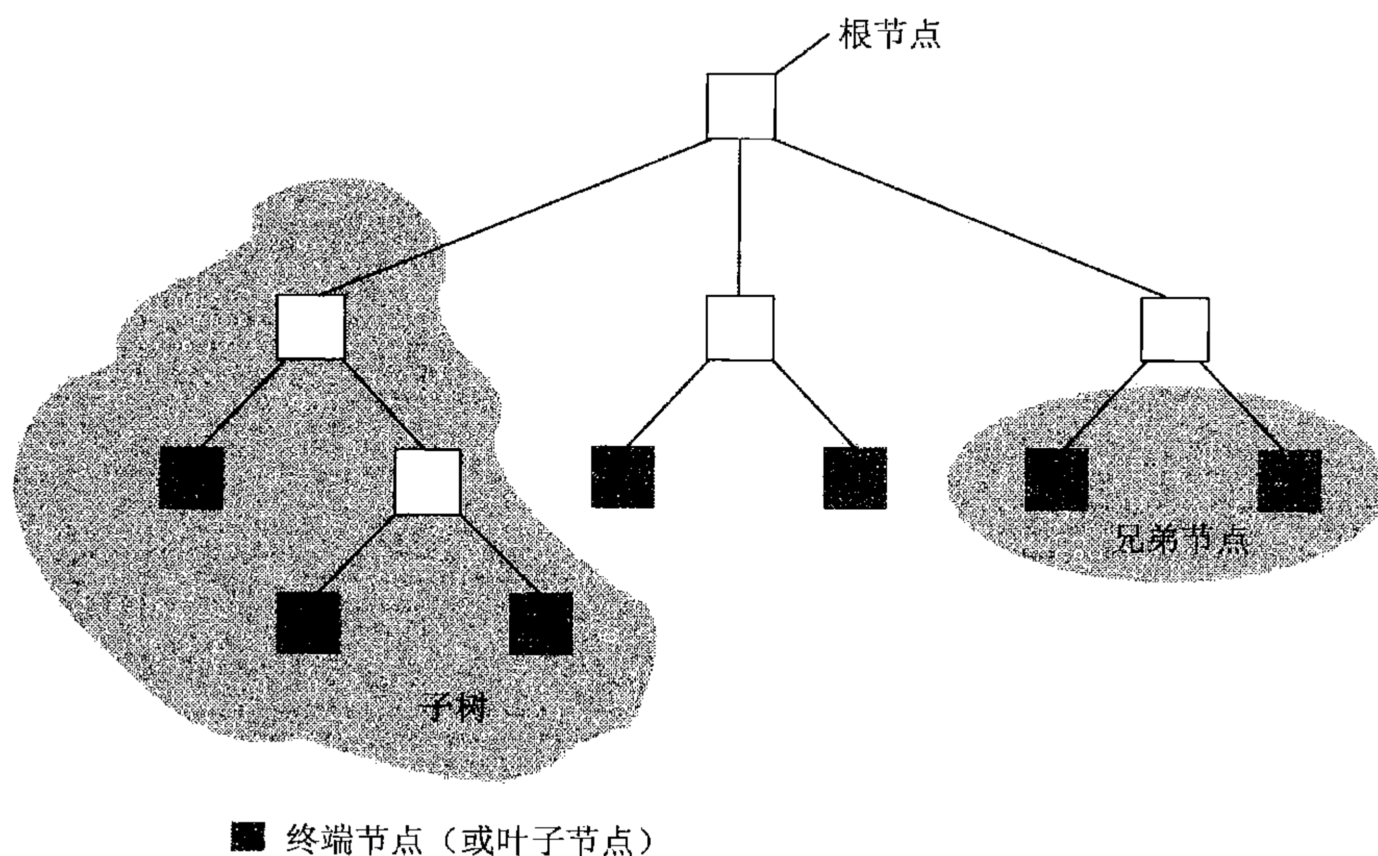


图8-3 树的术语

有时候，我们会提到这样一种树结构，该树的每个节点派生出其直接下层的节点。所以常常会说到一个节点的祖先和后代。这里，称一个节点的直接后代为**子**（children）节点，称其直接祖先为**父**（parent）节点。而将有同一个父节点的那些节点称之为**兄弟**（sibling）节点。如果一个树的每个父节点有不多于两个的子节点，那么称该树为**二叉树**（binary tree）。

如果选择一棵树中的任意一个节点，该节点与其下层的那些节点也构成了一个树结构，那么就称这些较小的结构为**子树**（subtree）。这样一来，每个子节点就是其父节点下面的子树的根节点，这样的子树称为父节点的一个**分支**（branch）。在二叉树中，提到树的显示方式时我们经

常会谈到，一个节点的左子树和右子树。

问题与练习

1. 针对以下每一个结构：列表、栈、队列和树，举出例子（计算机科学以外的）。
2. 总结出列表、栈及队列间的区别。
3. 假设A字母被放入一个空栈中，然后依次是字母B和C，再假设一个字母出栈，字母D和E入栈。请将栈中字母按照出现的自顶向下的顺序排列出来，如果一个字母要出栈，哪个字母将被检索？
4. 假设字母A放入一个空的队列中，然后依次是字母B和C。再假设此队列中一个字母被移出，之后插入字母D和E。请按照字母在队列中从表头到表尾出现的顺序列出它们。如果此时再要从队列中移出一个字母，应该是哪个字母？
5. 假设一个树有4个节点：A、B、C和D。如果A和C是兄弟，而D的父节点是A，哪些节点是叶子节点？哪些节点是根节点？

8.2 相关概念

在本节中，我们分别讨论3个与数据结构紧密相关的主题：抽象、静态与动态结构间的区别以及指针的概念。

8.2.1 抽象

前面小节中出现的数据结构常与计算机内存中所存储的数据有关。但是计算机的内存并不是按照数组、列表、栈、队列和树这样的结构来组织的，而是顺序地组织成一组可寻址的存储单元。这样一来，所有的其他结构都必须进行模拟。如何完成这种模拟工作是本章的主题。到现在为止，我们只是指出，数组、列表、栈、队列和树这样的组织都是些抽象工具，之所以构造这些抽象工具，是为了使数据的用户不用关心实际数据存储的细节，这样信息就好像是以一种更为方便的形式进行存储的，便于用户访问。

在这里，*用户*这个术语并不一定指人，这个词的含义因视角和时间而异。如果从一个使用PC机来维护保龄球比赛记录的人的角度考虑，那么用户就是一个人。在这种情况下，应用软件（也许是电子制表软件包）将负责把数据用人觉得方便访问的抽象形式表示出来（很可能用同构数组来表示）。如果从因特网上的一个服务器的角度考虑，那么这时的用户可以是一个客户端。在这种情况下，服务器将负责把数据表示成便于客户端访问的抽象形式。如果从程序的模块结构来考虑，那么用户应该是需要访问这些数据的任何模块。在这种情况下，模块所包含的数据应该负责把数据表示成便于其他模块访问的抽象形式。所有这些情况中，有一条共同的主线，那就是用户拥有将数据作为一个抽象工具来访问的特权。

8.2.2 静态结构与动态结构

构建抽象数据结构中的一个重要区别是：所模拟的结构是静态的还是动态的。也就是说，结构的形状或大小是否会随时间改变。例如，如果这个抽象工具是一个名字列表，那么考虑以下情况将非常重要：这份名字列表是会一直保持固定的大小，还是可能因名字的增加和删除而扩大和缩小。

就一般规律而言，静态结构比动态结构更容易处理。如果一个结构是静态的，那么仅仅需要提供一种能够访问结构中不同项的方法就可以了，也许就是能改变指定位置的数据值的方法。但是，如果结构是动态的，那就必须要处理增加和删除项的问题，还要找到因数据结构增长所

需的存储空间。在结构设计不合理的情况下，增加一个新项可能会导致对结构进行大规模的重排，而且结构的过度增长可能会迫使整个结构转移到另一个可用空间更大的存储区域。

8.2.3 指针

我们知道计算机内存中各种不同的单元是由数字地址来标识的。作为数值，这些地址本身就可以进行编码，存放在内存单元中。**指针**（pointer）是一个存储区，包含了这样的被编码过的地址。对于数据结构，指针用来记录数据项存放的位置。例如，如果我们必须要不断地将一个数据项从一个位置移到另一个位置，那么可以指定一个固定的位置，将其作为指针。这样一来，每次移动该项时，就能够通过更新这个指针来反映数据的新地址。接下来，当要访问该数据项时，就可以通过指针来找到该项。事实上，指针将一直指向数据。

在第2章学习CPU的过程中，已经遇到过指针这个概念。在那一章中，我们可以看到，称为程序计数器的寄存器被用来存放下一条要执行的指令的地址。所以，程序计数器就起到了指针的作用。事实上，程序计数器的另一个名字叫作**指令指针**（instruction pointer）。

举一个指针应用的例子，假设在计算机内存中，按书目的字母顺序存放着小说的清单。虽然在许多应用中，这样的安排比较方便，但是如果要寻找某个作者的所有小说作品就比较困难了，因为它们分散在整个列表中。为了解决这个问题，可以在表示每本小说的存储单元块中保留一个额外的存储单元，并将该存储单元用作一个指针，指向表示同一作者另一本小说的存储块。通过这种方法，同一作者的所有小说就可以链接成一个环（如图8-4所示）。找到给定作者的一本小说以后，我们就可以循着指针一本接另一本地找到该作者的所有小说。

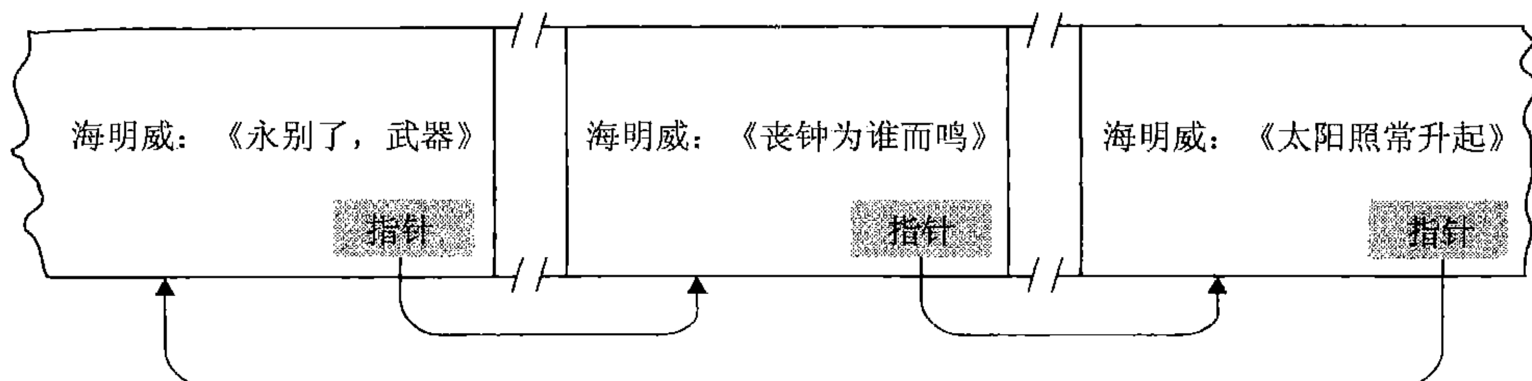


图8-4 按书名排列而根据作者链接的小说

现代的许多程序设计语言都把指针作为一种基本的数据类型。也就是说，就像对整数、字符串那样，程序设计语言也可对指针进行声明、分配以及操作。利用这种语言，程序员就能在计算机存储器中，把相关的项用指针相互链接起来，从而设计出精巧的数据网。

问题与练习

1. 数组、列表、栈、队列和树等数据结构在何种意义上是抽象的？
2. 请举出一个涉及静态数据结构应用的例子。再举出一个涉及动态数据结构应用的例子。
3. 请举出在计算机科学领域外出现指针这个概念的例子。

8.3 数据结构的实现

现在我们来讨论8.1节所介绍的一些数据结构在计算机内存中的存储方式。

8.3.1 数组的存储

我们首先讨论存储数组的技术。正如第6章所介绍的，在高级程序设计语言中，常常将这些

结构作为基本结构来提供。在此，我们的目标就是要理解如何将处理这些结构的程序翻译成用来处理存放在内存中的数据的机器语言程序。

1. 同构数组

假设要存储一个24小时温度的读数序列，每个读数需要存储空间的一个存储单元。而且，假设依据它们在序列中的位置来确定这些读数。也就是说，我们要能够访问第1个读数或者是第5个读数。简单来说，就是要按照一维同构数组的方式来处理这个序列。

这里，只要将这些读数按顺序存放在具有连续地址的24个存储单元中，就可以实现这个目标了。那么如果这个序列中第1个单元的地址是x，那么任何一个指定温度读数可以这样计算得到，即用所要读数的序号减去1，然后将计算的结果加上x。具体来说，第4个读数就放在 $x+(4-1)$ 这个地址中，如图8-5所示。

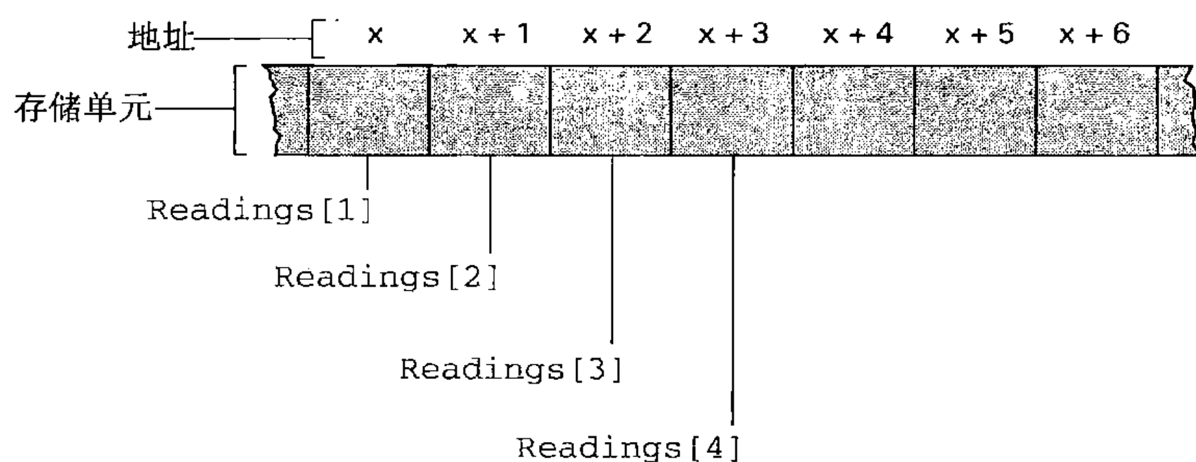


图8-5 存放在存储器中的温度读数数组，起始地址为x

这种技术为大多数高级程序设计语言的翻译程序所采用，用以实现一维同构数组。当翻译程序遇到下面这样的声明语句时：

```
int Readings[24];
```

这就表明，`Readings`这个术语是指可以存放24个整数的一维数组，这时，翻译程序就会安排预留24个连续的存储单元。以后在程序中，如果遇到赋值语句

```
Readings[4] ← 67;
```

则要求将值67放入数组`Readings`的第4项中。此时，翻译程序就生成了一串机器指令，把值67放入地址为 $x+(4-1)$ 的存储单元中，其中x为与数组`Readings`相关的存储块中第一个单元的地址。通过这种方式，程序员在编写程序的时候，就可以认为温度读数确实存放在一个一维数组中。（注意，在C、C++、C#及Java语言中，数组的下标是从0而不是从1开始的，这样一来，第4个读数应该由`Readings[3]`表示。见本节末的问题与练习3。）

现在，假设我们要记录一个公司的销售人员一周内的销售业绩。在这种情况下，可以想象将数据组织成一个二维同构数组。在该数组中，每行的值表示某个员工的销售业绩，而每列中的值表示某一天内所有的销售业绩。

为了实现这种要求，首先要认识到这个数组是静态的，即使它的内容得到更新，其大小也不会改变。所以就可以计算出存放整个数组所需的存储区的总数，然后就保留这样大小的一块连续存储单元。接下来就一行一行地把数据存入数组，从所保留的存储块的第1个存储单元起，把数组第1行数值存进连续的存储单元；接着存放下一行，再下一行，以此类推（如图8-6所示）。这样一种存储系统称为**行主序**（row major order）系统。与之相反的是，如果数值一列接着一列地存放，则称**列主序**（column major order）系统。

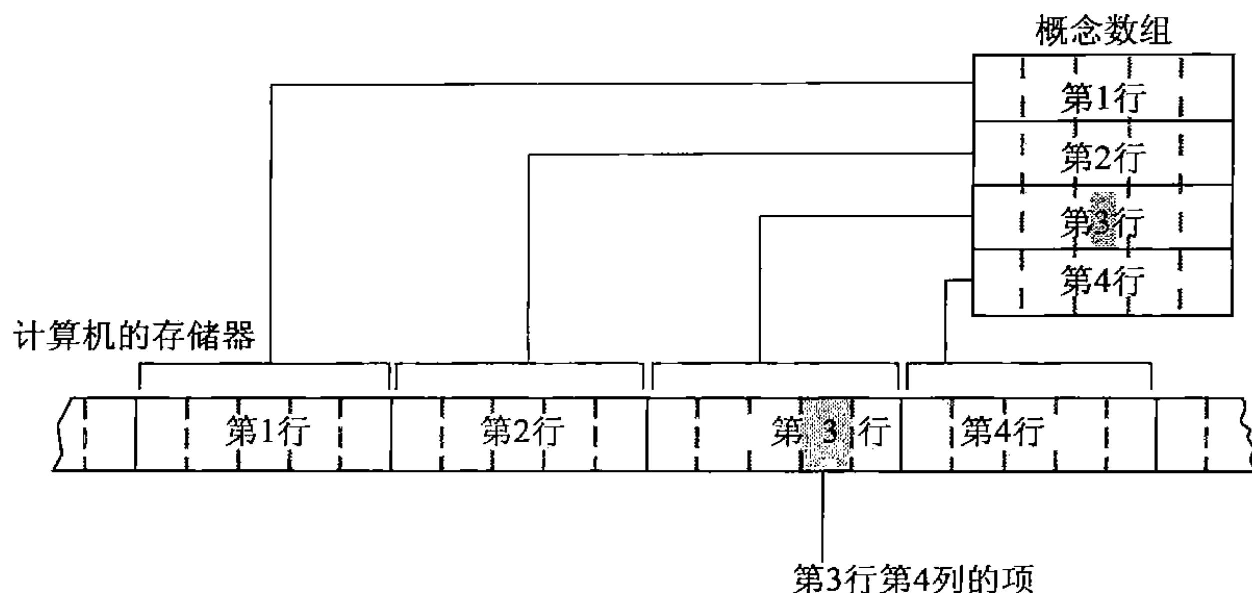


图8-6 以行主序方式存储的一个4行5列二维数组

如果数据以这种方式存放，那么考虑一下，如何找到数组中第3行第4列的数值？设想一下，我们处在所保留的机器存储块的第1个单元。从这个位置起，可以依次找到数组第1行的数据，接着是第2行，然后是第3行，依次类推。要得到第3行的数据，我们必须先经过第1行和第2行。由于每一行有5个项（星期一至星期五，每天一个项），因此要访问到第3行的第一个项，必须经过一共10个项。从那起，我们还必须再经过3个项，才能到达第3行第4列的那个项。这样，为了到达第3行第4列的项，从存储块的开始处总共需要经过13个项。

上述的计算过程可以概括为一个公式，即可以将行列位置的索引转换为实际的存储器地址。具体来说，如果令c表示一个数组的列数（也就是每行所包含的项的个数），那么第i行第j列项的地址就可以表示为：

$$x+(c\times(i-1))+(j-1)$$

其中，x是第1行第1列项的单元地址。也就是说，必须经过i-1行（每行包括c个元素），才能到达第i行，然后再经过j-1个项，才能到达这行的第j项。上面的例子中，c=5，i=3，j=4，所以，如果数组从地址x进行存放，那么第3行第4列的项的地址就应该为$x+(5\times(3-1))+(4-1)=x+13$。表达式$(c\times(i-1))+(j-1)$有时候称为**地址多项式**（address polynomial）。

这也是大多数高级程序设计语言的翻译程序所采用的技术。当遇到声明语句

```
int Sales[8,5];
```

时，则表明：Sales是一个8行5列的二维整数数组，翻译程序就会留出40个连续的存储单元。以后如果遇到赋值语句

```
Sales[3,4] ← 5;
```

则需要将数值5放到数组Sales的第3行第4列的那个项中，此时，就产生一串机器指令，将数值5放到地址为$x+(5\times(3-1))+(4-1)$的存储单元中，其中，x是与数组Sales相关联的存储块的第1个单元的地址。通过这种方式，程序员编写程序时，就好像销售量确实存放在一个二维数组中一样。

2. 异构数组

现在，假设要存储的异构数组称为Employee，该数组包含3个部件：Name（字符型），Age（整型），SkillRating（实型）。如果数组中的每个部件所需的存储单元的数目是固定的，那么就可以将数组存放在一个连续的单元块中。例如，假设Name部件最多需要25个单元，Age只需要一个单元，SkillRating也只需一个单元。于是，我们就可以预留出一个有27个连续单元的存储块，开始的25个存储单元用来存放员工的名字，第26个存储单元用来存放员工的年龄，最后一个存储单元用来存放员工的技能等级（如图8-7a所示）。

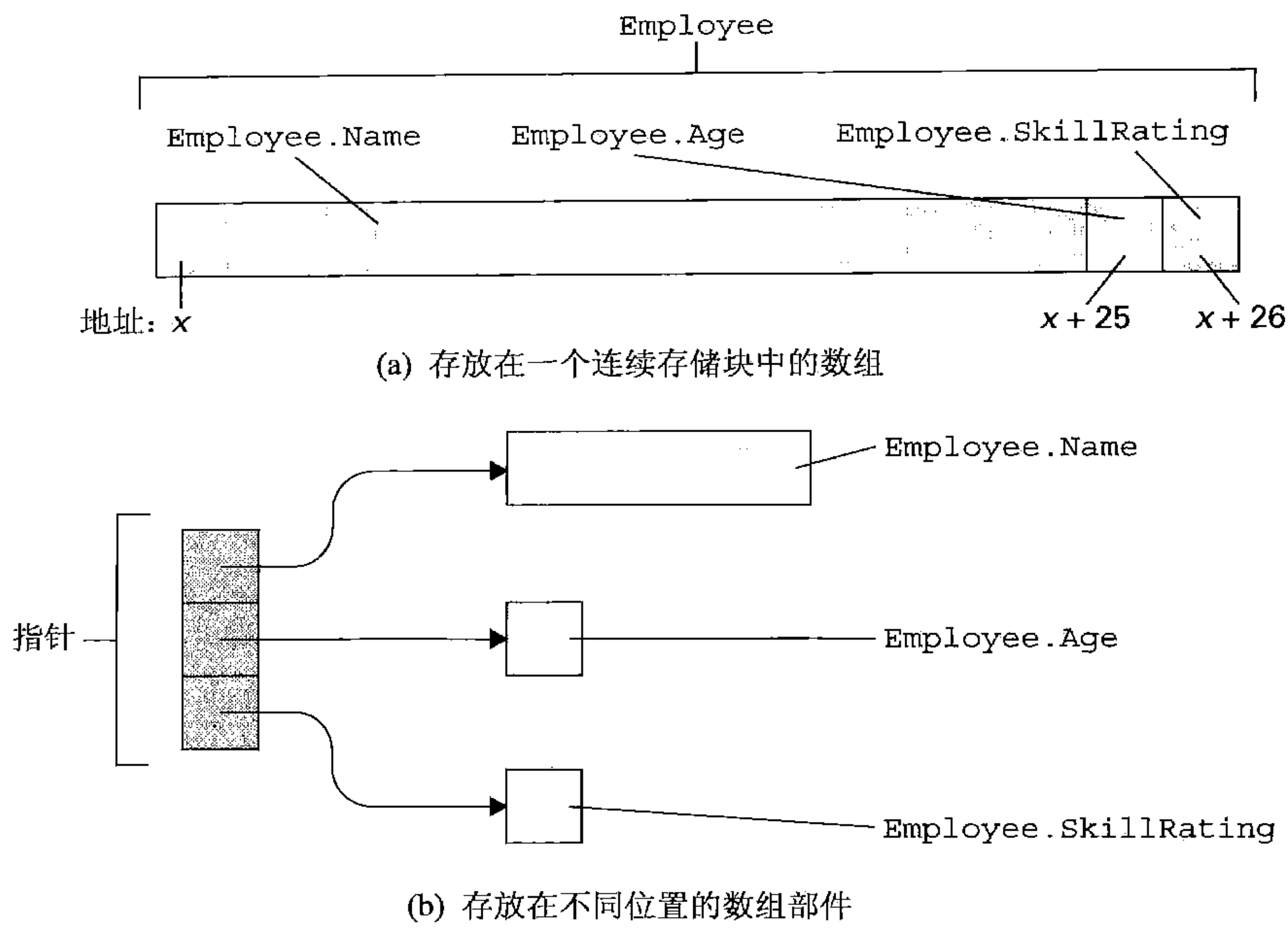


图8-7 存储同构数组Employee

通过这种安排，就可以很容易地访问该数组中不同的部件。例如，如果第一个存储单元的地址是x，那么指向Employee.Name（意思是Employee数组中的Name部件）的任何引用都将转移到从地址x开始的25个存储单元，而指向Employee.Age（意思是Employee数组中的Age部件）的引用将转移到地址为$x+25$的存储单元。具体来说，如果翻译程序遇到了高级语言中的这样一条语句：

```
Employee.Age ← 22;
```

那么只要产生一系列机器语言指令，用以将数值22放入地址为$x+25$的存储单元。或者说，如果将EmployeeOfMonth定义为一个与之类似的数组，并将其存放在地址为y的存储块上，那么语句

```
EmployeeOfMonth ← Employee;
```

将会翻译成一个指令序列，将起始地址为x的27个存储单元的内容复制到起始地址为y的27个存储单元中。

在一个连续存储单元块中存储异构数组的另一种方法就是，将异构数组的每个部件分别存放在不同的位置，然后通过指针的方式将它们链接在一起。更准确地说，如果这个数组包含有3个部件，那么就在存储器中找到一个位置，用以存放3个指针，每个指针指向一个部件（如图8-7b所示）。如果这些指针存放在以x为起始地址的存储块中，那么通过存放在地址为x处的指针就可以找到第1个部件，通过存放在地址$x+1$处的指针就可以找到第2个部件，以此类推。

这种存储方式在有些场合尤其适用，如数组部件的大小是动态的情况。举例来说，利用这种指针系统，只需要在存储器中找到一个存储区来存放较大的部件，然后调整相关的指针，令其指向这个新位置，这样就可以增加第一个部件的大小了。但是，如果数组是存放在一个连续的存储块中，那么不得不修改整个数组。

8.3.2 列表的存储

现在来讨论将一个名字列表存放在计算机内存中的技术。一种方法就是将整个列表存入具

有连续地址的一整块存储单元中。假定每个名字不超过8个字母，我们可以把这个大的整块存储单元分成一组子块，每个子块包含有8个存储单元。每个子块中放入一个用ASCII码记录的名字，一个单元放入一个字母。如果一个名字不足填满分配给子块的所有存储单元，只需用空格的ASCII码将剩余的单元填满就行。利用这种方式，存放一个10个名字的列表需要一个有80个连续单元的存储块。

图8-8所概括的就是这种存储系统。其重点就在于，整个列表都存储在一大块内存中，其连续的项依次存放在相邻的存储单元中。这样的一种组织称为**邻接表**（contiguous list）。

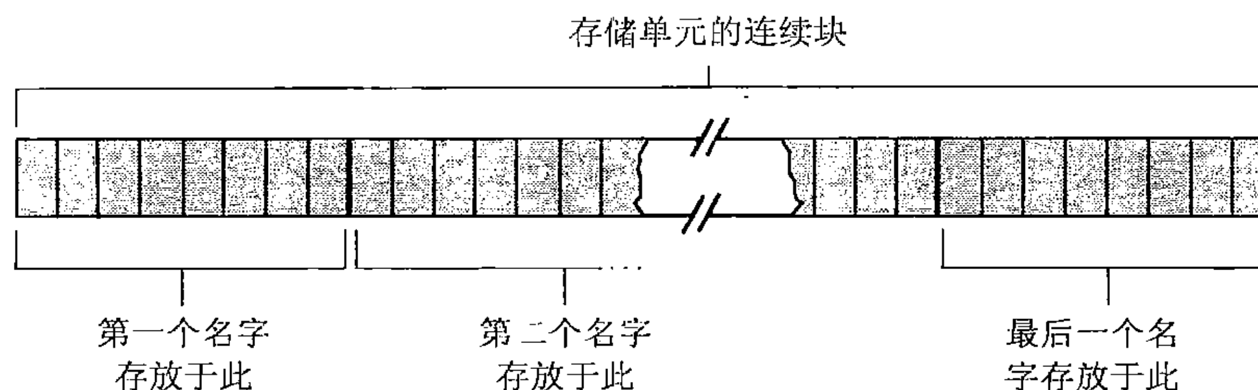


图8-8　名单作为一个邻接表存放在内存中

邻接表的实现

大多数高级程序设计语言都提供了构建和操作数组的原语，它们都是构建和操作邻接表的方便工具。如果表的项都是相同的基本数据类型，那么该表就是一个一维同构数组。稍微复杂的例子是文中所讨论过的一张有10个名字的名单，每个名字不超过8个字符。这种情况下，程序员可以构建这样一个邻接表，即为一个10行8列的二维字符数组。该表可以用图8-6所表示的结构来表示（假设该数组以行主序进行存放）。

许多高级语言都含有支持这种表实现的功能。例如，假设将上面所提到的二维字符数组称为`MemberList`，那么依据传统的表示法，表达式`MemberList[3,5]`就是指第3行第5列的那个字符。有些语言用表达式`MemberList[3]`来指整个第3行，也就是表中的第3个项。

邻接表这种存储结构用来实现静态表很方便。但就动态表的情况而言，则有些不便之处，因为名字的添加和删除都会导致不断地对项进行移位操作，这样就比较耗时。最坏的情况下，项的增加甚至会导致这样一个问题：为了能够获得足够大的存储单元来存放这个扩展过的列表，必须把整个表移到一个新的位置。

如果一个表中的各个项不必一起存放在连续的大块存储区中，而可以各自存放在不同的存储区域，那么这些问题就可以得到化解。为了说明这个问题，仍然考虑存放名单的例子（每个名字不超过8个字母）。这次，将每个名字存放在一个有9个连续存储单元的块中。前面8个存储单元用来存放名字本身，最后一个单元用作指针，指向表中的下一个名字。遵循这种方法，整个表可以分散在若干个较小的由指针链接起来的9单元块中。由于这种链接系统，就将这样一种数据的组织方式称为**链表**（linked list）。

为了记下链表的起始点，我们另外再设一个指针，用来存放第一个项的地址。由于这个指针是指向链表的起始点，或者叫头节点，所以将此指针称为**头指针**（head pointer）。

为了标记链表的结束，我们使用了**NIL指针**（NIL pointer），也称为**空指针**（NULL pointer），这只是放在最后一项的指针单元中的一个特殊的位模式，用来表示链表中不会再有别的项。例如，如果约定不会在0地址存放表项，那么0值就不会成为合法的指针值，因而就可以将0值用作NIL指针。

最后的链表结构如图8-9所示，图中用几个单个的矩形表示存放链表的分散存储块。每个矩

形都标识出了它们的组成元素。指针用箭头来表示，从指针本身引向指针的被寻地址。如果要遍历整个链表，就需按照头指针找到第一个表项，由此出发，按照项中所存储的指针，一个接一个地进行遍历，直至遇到NIL指针。

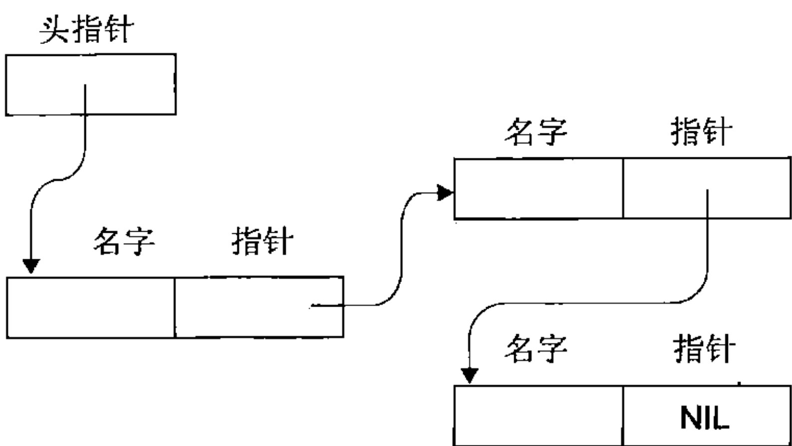


图8-9 链表的结构

为了说明链表相对于邻接表的优势，这里考虑删除一个项的操作。在邻接表中，删除一个项就会产生一个空缺，这就意味着，被删除项的后续项必须向前移动来保持表的连续。然而，在链表的情况中，删除一个项只需改变一个指针即可。也就是说，将原本指向被删除项的指针修改成指向被删除项后面的那个项（如图8-10所示）。这样一来，当遍历这个链表时，由于被删除项已不再是链的一部分，因而就会被忽略。

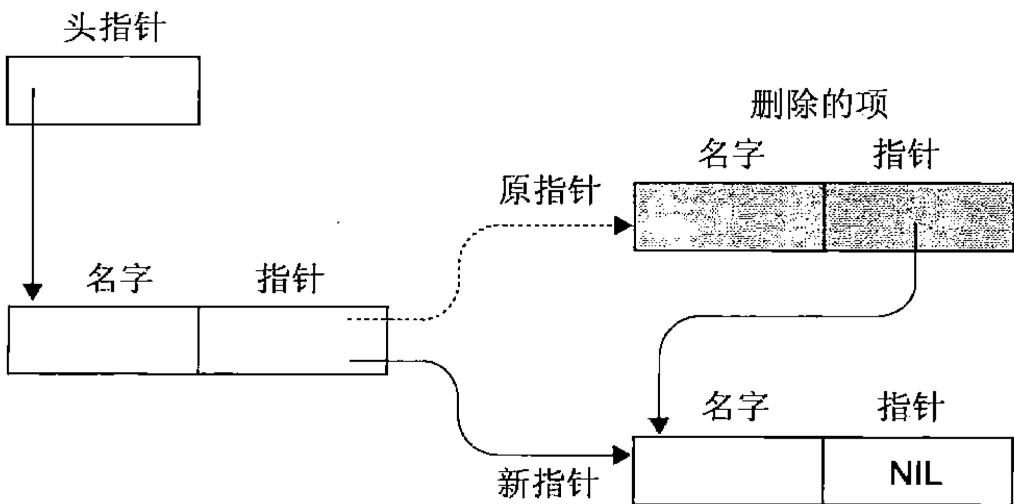


图8-10 从链表中删除一个项

指针的一个问题

就像使用流程图会导致杂乱的算法设计（参见第5章），以及随意使用goto语句会导致蹩脚的程序设计（参见第6章）一样，乱用指针也会产生不必要的复杂性和易错的数据结构。为了克服这种混乱，许多程序设计语言严格限制了指针的灵活性。例如，Java语言不支持一般形式的指针，而只支持称为引用的一种受限形式的指针。其区别在于，引用不能被算术运算修改。举例来说，如果Java程序员想把Next引用前移至邻接表中的下一项，那么将会用等价于下面这条语句的语句：

将Next重新定位到下一个表项

而C程序员则会用等价于下面这条语句的语句：

将Next的值赋给Next+1

注意，Java语句能更好地反映其根本目标，而且，为了执行这条Java语句，另一个表项必须存在。但是，如果Next已经指向了表中的最后一项，那么执行这条C语句，其结果将会造成Next指向表外的某个地方，这是C编程新手甚至经验丰富的老手经常犯的一个错误。

在链表中插入一个新项的工作就稍微麻烦一点。首先要找到一个能够容纳新项及其指针的未用存储块，然后将该项存入，并将该项的指针域填为应接在新项后面的那个项的地址。最后，修改该新项的前一项的指针，令其指向新项（如图8-11所示）。做了这样的修改后，每次遍历链表的时候，就能在合适的位置遍历到新项。

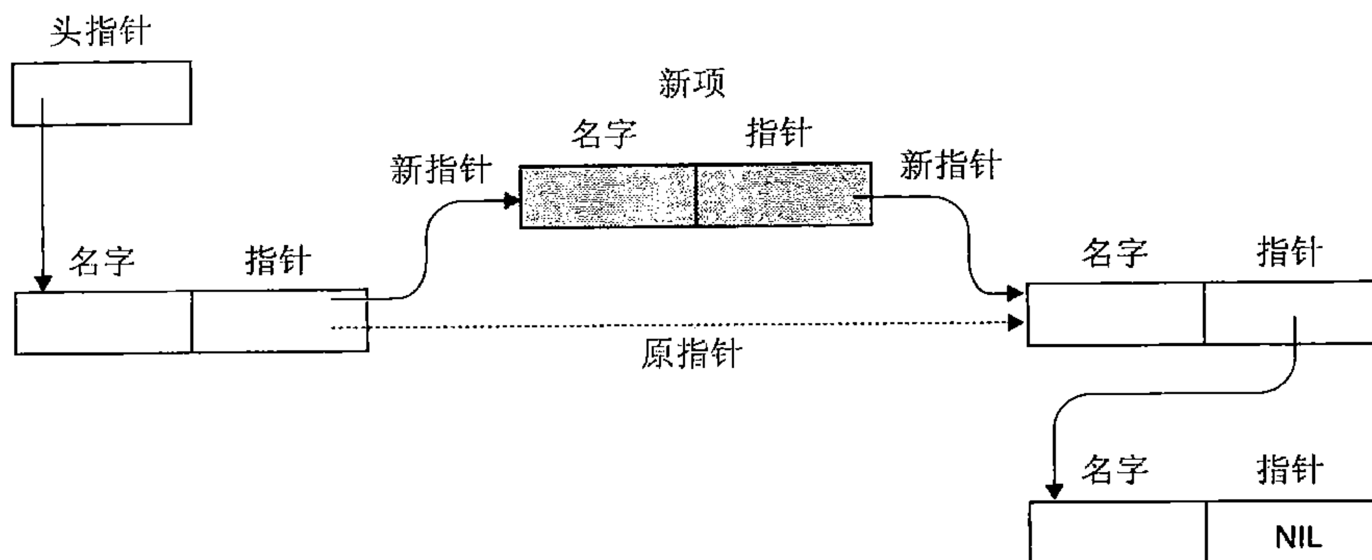


图8-11　向链表中插入一个项

8.3.3　栈和队列的存储

为了存储栈和队列，通常采用一种类似于邻接表的存储方式。就栈而言，所预留的存储块，其大小要足够容纳其大小达到最大时的栈。（确定这个存储块的大小，往往很关键。如果预留的空间太少，则栈可能会超出所分配的存储空间；然而，如果预留的空间太多，则会浪费存储空间。）将这个存储块的一端指定为栈底，压入栈的第一个项就存储在这里。于是，再入栈的项就放在其上一个入栈的项的旁边，这样一来，栈就向着预留块的另一端生长。

注意到，在项入栈和出栈时，栈顶的位置会在预留的存储块中来回移动。为了跟踪这个位置，就用一个外加的存储单元来存放栈顶的地址，这个存储单元称为**栈指针**（stack pointer）。也就是说，栈指针是指向栈顶的指针。

如图8-12所示，整个栈系统是这样工作的：为了向栈中压入一个新项，首先要调整栈指针，令其指向正好在栈顶上边的空闲处，然后将新项存放在这个位置。为了从栈顶弹出一个项，先读取栈指针指向的那个数据，然后调整栈指针，令其指向栈中的下一个项。

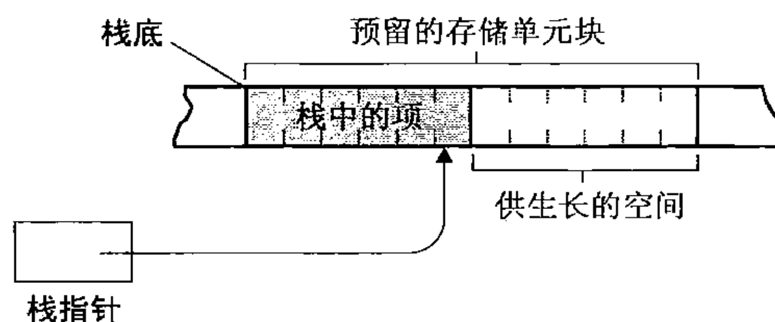


图8-12　存储器中的一个栈

队列的传统实现方法类似于栈的实现方法，也是在内存中预留一块连续的存储单元，其大小要足够容纳其大小达到最大时的队列。然而，就队列而言，需要在队列的两端都进行操作，所以不能像栈那样只用一个指针，这里需预留两个存储单元用作指针。一个指针称为**头指针**（head pointer），用来跟踪队列的头；另一个指针称为**尾指针**（tail pointer），用来跟踪队列的尾。当队列为空时，这两个指针指向同一个位置（如图8-13所示）。每当一个项进入队列时，就将该项放在由尾指针指向的位置，然后修改尾指针，令其指向下一个空闲的位置。通过这种方式，尾指针始终指向队列尾部的第一个空闲位。如果要从队列中删除一个项，则先读取头指针指向的那个项，然后调整头指针，令其指向队列中的下一项。

至此，所描述的这种存储系统存在一个问题，即随着项的插入和删除，队列会像冰川一样在内存中缓慢移动（如图8-13所示）。这样一来，就需要一种机制，使队列保持在预留的存储块中。这个问题的解决办法很简单，就是让队列自始至终都在所分配的存储块中移动。于是，当队尾到达队列的末端时，如果再增加一个项，则回到存储块的起始端进行增加操作，这时的起始端是空闲的。同样，当存储块中最后一个项成为队首并被删除时，就将头指针调整为存储块的起始端，这时，新项等着进入队列。在这种方式下，队列在存储块内按环状依次排列，仿佛存储块的尾部被连结在一起，形成了一个循环（如图8-14所示）。这种技术所实现的队列称为**循环队列**（circular queue）。

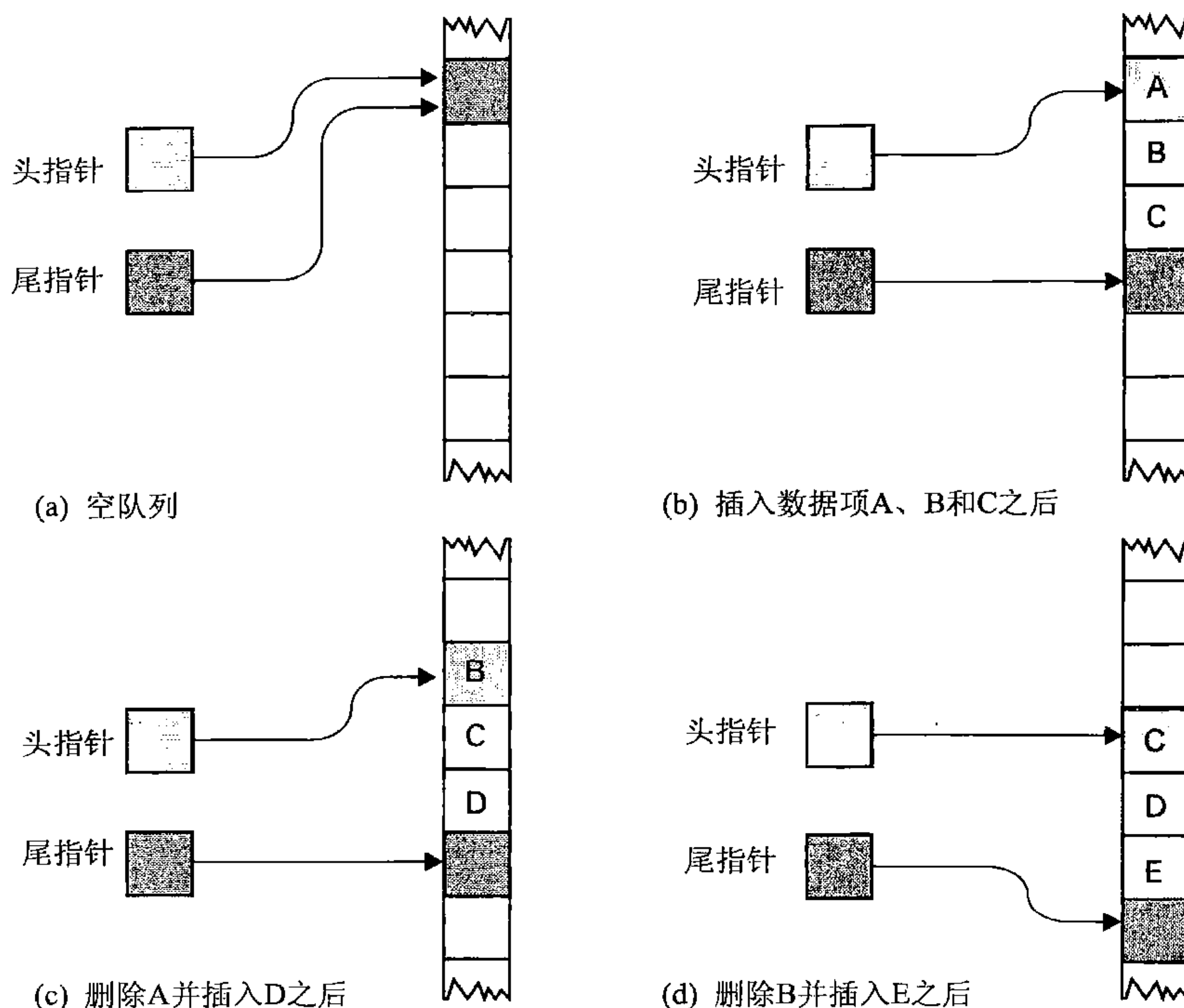


图8-13 用头指针和尾指针实现一个队列。注意看随着项的插入和删除，队列是怎样在内存中移动的

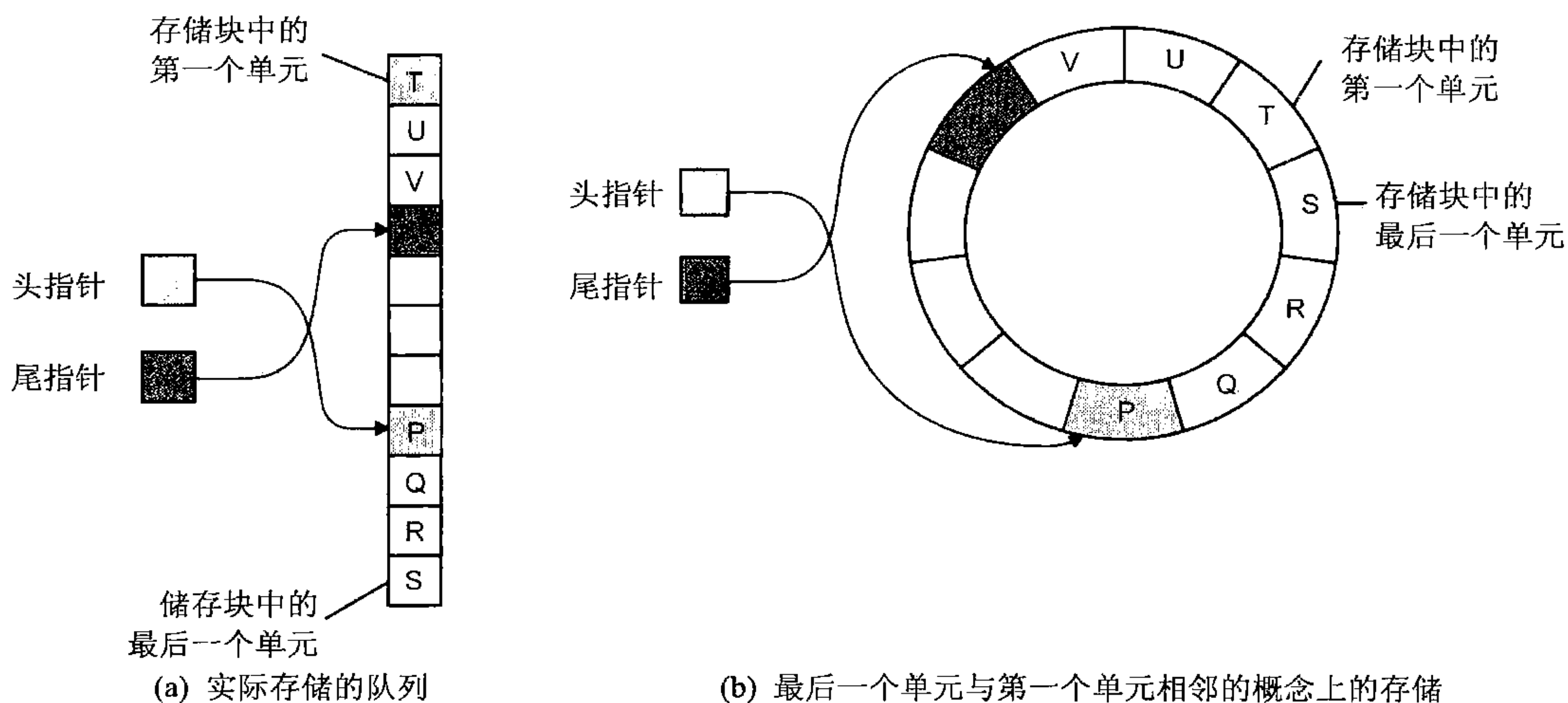


图8-14 包含字母P到字母V的循环队列

8.3.4 二叉树的存储

关于树的存储技术，这里将注意力限于讨论二叉树。我们说过，二叉树的每个节点至多只有两个子节点，它通常采用类似于链表所用的链接结构存放在存储器中。然而，二叉树的每个项（或称为节点）不是由两个元素组成（数据及指向下一个节点的指针），而是由三个元素组成：（1）数据；（2）指向该节点的第一个子节点的指针；（3）指向该节点的第二个子节点的指针。尽管在计算机中，不存在左右之分，但这里为了方便，就将第一个指针称为**左子指针**（left child pointer），而另一个指针称为**右子指针**（right child pointer），这样就可以方便地在纸上

画出树的图形。所以，二叉树的每个节点可由一个简短的、连续的存储块来表示，其格式如图8-15所示。

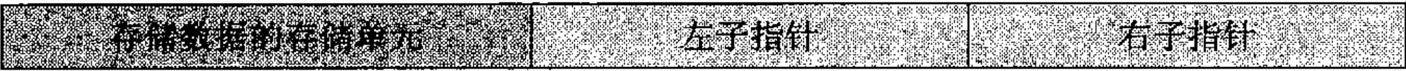
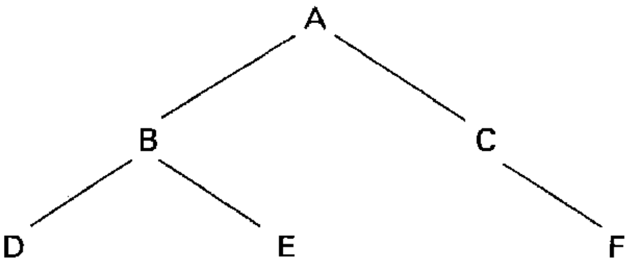


图8-15 二叉树中的一个节点的结构

要在存储器中存储树，首先要找到可用的存储单元块来存放树节点，然后依据所要求的树结构，将这些节点链接起来。每个指针必须设置成指向其相应节点的左右子节点，如果树的某个方向不再有节点，则将相应的指针赋值为NIL。（这也就说明，终端节点的特征就是其两个方向的指针值都为NIL。）最后，留出一个专门的位置，称为**根指针**（root pointer），用来存放根节点的地址。对树的访问就是从根指针开始的。

图8-16所示的就是这样一种链接存储系统的例子，在该图中，既画出了一个二叉树的概念结构，又展示出了该树在计算机存储器中实际的存储形式。可以看出，树的节点在主存中的实际组织形式与概念上的组织形式有很大的不同。然而，沿着根指针就能找到根节点，然后随着相应的指针，从一个节点到另一个节点，由上至下地遍历树。

概念树



实际的存储结构

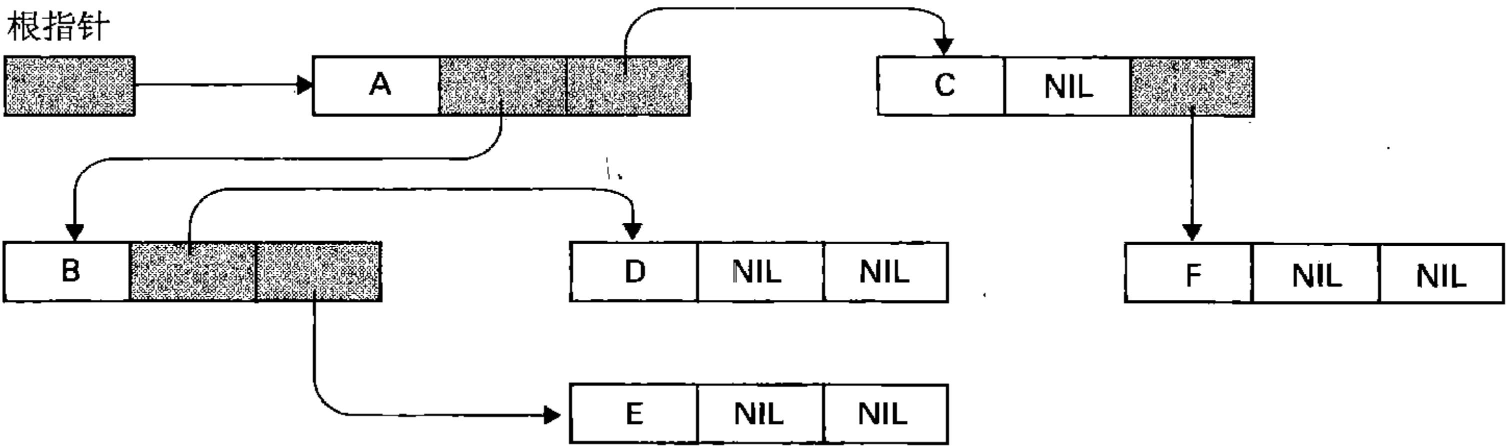


图8-16 二叉树的概念组织和使用链接存储系统实现的实际组织形式

对于二叉树的存储，除了用链式结构外，还可以用单个的、连续存储单元块来存放整个树。利用这种方法，把树的根节点存放在这个存储块的第1个单元。（为了简单起见，假设树的每个节点只需要一个存储单元。）然后，把根的左子存放在第2个单元，根的右子存放在第3个单元。就一般情况而言，单元n中节点的左、右子节点分别存放在单元$2n$和$2n+1$中。在存储块中，没有被树用到的单元就用一个特别的位模式来标识，表示这个位置没有数据。利用这种技术，图8-16中所示的树可以像图8-17所示那样进行存储。注意，这种存储系统实际上是由上至下地将树中各层的节点作为存储段来存放，一层接着一层。也就是说，存储块中的第1个项是根节点，接下来是根节点的子节点，然后是孙子节点，以此类推。

与前面所描述的链接结构不同，这种存储系统在查找某个节点的父节点或兄弟节点方面就显得更为有效。一个节点的父节点的位置可以这样确定：将该节点的位置除2，然后丢掉余数即得（如位置为7的节点，其父节点的位置为3）。一个节点的兄弟节点的位置可以这样确定：如果

该节点的位置为偶数，则其兄弟节点的位置加上1即得；如果该节点的位置为奇数，则其兄弟节点的位置减去1即得。例如，位置为4的节点，其兄弟节点的位置为5；位置为3的节点，其兄弟节点的位置为2。而且，当二叉树接近平衡（也就是说，根节点下的两个子树具有同样的深度）和完全平衡（也就说，该树没有瘦长的分支）的情况下，这种存储系统对存储空间的利用更为有效。不过，对于不具备这些特征的树，该系统的效率就会很低，如图8-18所示。

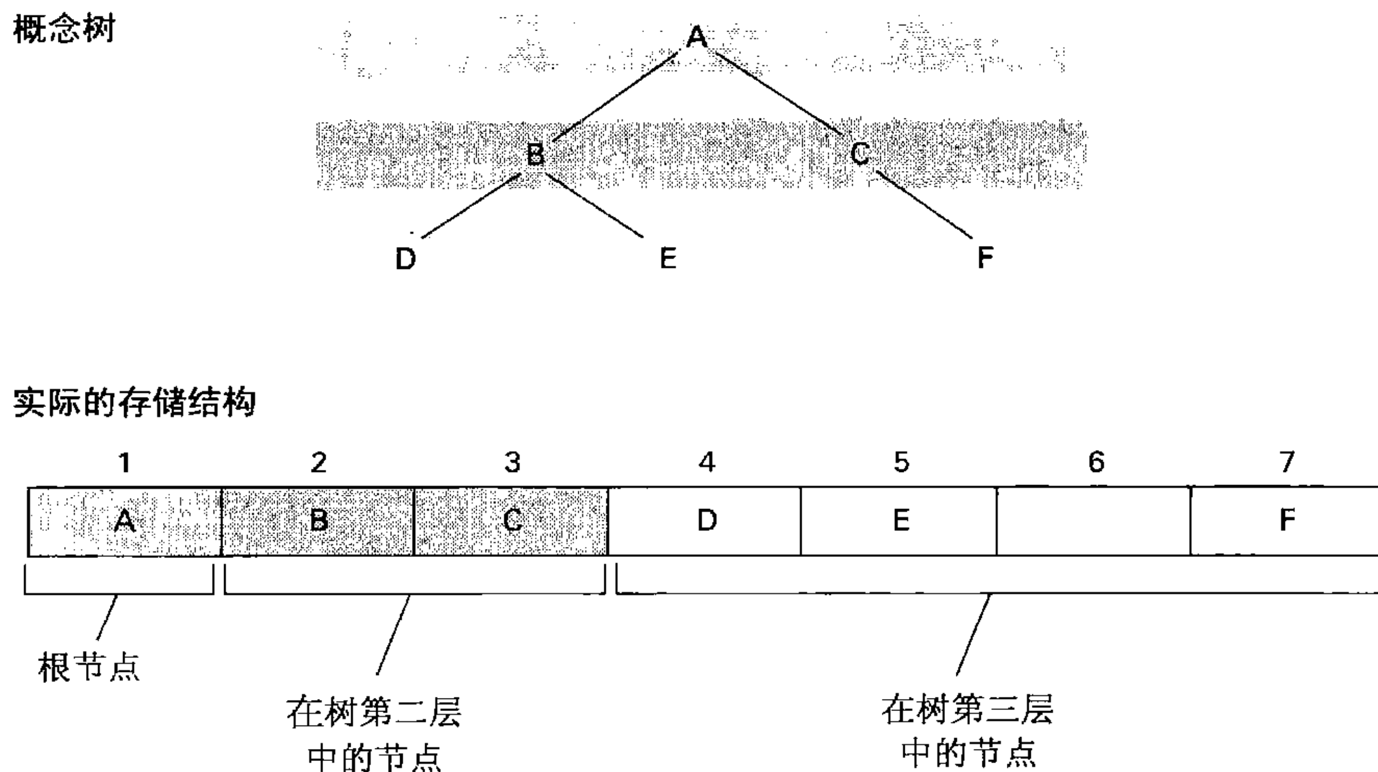


图8-17 指针存储的一棵树

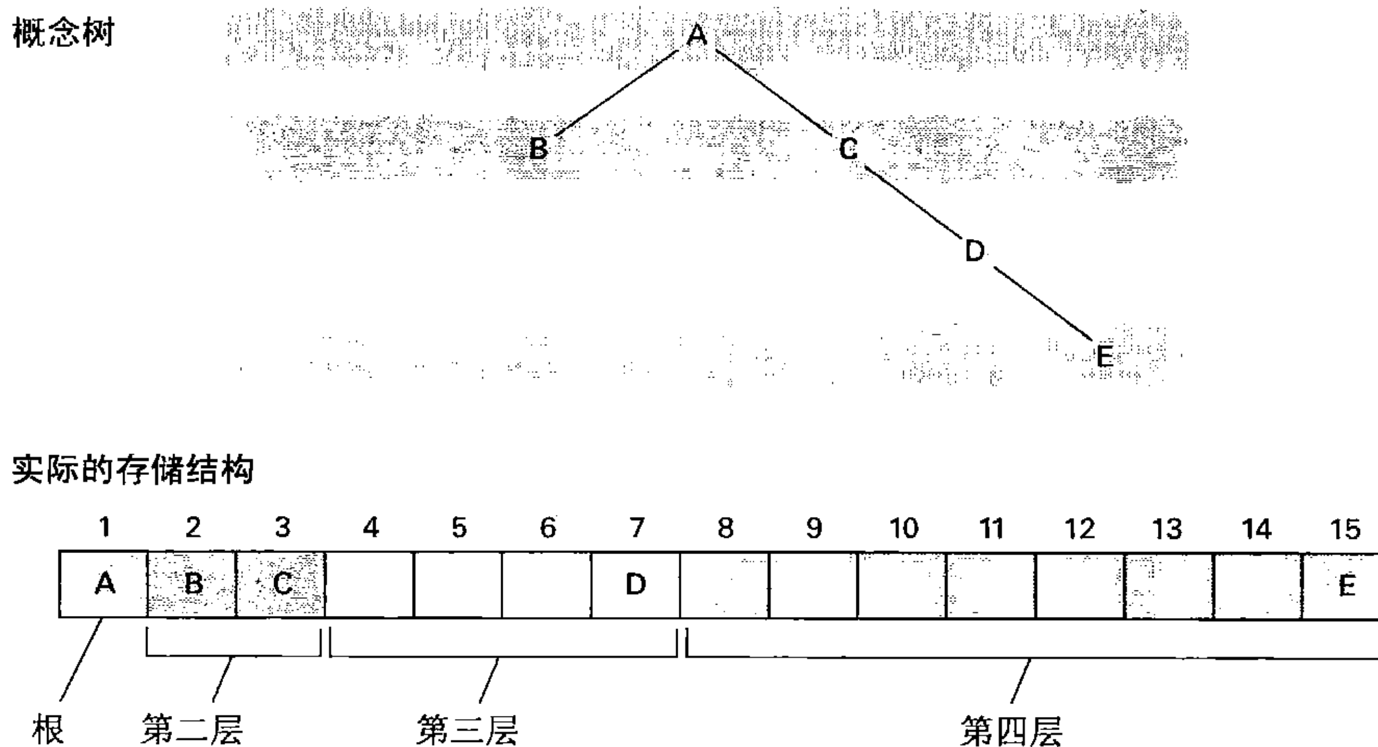


图8-18 一个稀疏不平衡树的概念形式以及不用指针的存储方式

8.3.5 数据结构的操作

我们已经看到，数据结构在计算机内存中的实际存储方式与用户想象的概念结构是不同的。二维同构数组实际上并不是存放在一个二维的矩形存储块中，表或树实际上可能由分散在较大范围的存储区域内的小片段组成。

所以，为了让用户将数据结构作为一种抽象工具来访问，必须对用户屏蔽实际存储系统的复杂性。这就意味着，用户所给出的指令（按照抽象工具的方式规定的）必须翻译成适合实际存储系统操作的步骤。对于同构数组而言，我们已经看到，翻译程序如何利用地址多项式将行、

列下标转化成存储单元地址。具体来说，语句

```
Sales[3,4] ← 5;
```

在程序员编写时，只将其作为抽象的同构数组来考虑，而我们已经知道，如何将这条语句转化为完成对主存进行正确修改的操作步骤。同样，我们也知道，涉及抽象异构数组的语句

```
Employee.Age ← 22;
```

如何依据该数组的实际存储情况将其翻译成合适的操作。

在表、栈、队列以及树这些情况中，根据抽象结构所定义的指令通常是利用过程将其转化为相应的操作的，而这些过程为用户屏蔽底层存储系统的细节的同时，还完成其预期的任务。例如，如果insert过程是用来在链表中插入新项，那么只要执行如下的一个过程调用，就可以将J. W. Brown加入Physics 208班的学生名单中：

```
insert (" Brown,J.W.",Physics208)
```

可以注意到，这个过程调用完全是依据抽象结构声明的，通过这种方式，可以把表的实际执行过程隐藏起来。

图8-19所示的是一个更为详细的例子，名为printList的过程被用来打印名字链表。在这个过程中，假设头指针指向了链表中的第一个项，而每个项都由两个元素组成：名字和指向下一个项的指针。这个过程编写好以后，就可以作为一个抽象工具用来打印链表，而无需关心打印链表所需的实际步骤。例如，要获得一份Economic 301班打印的学生名单，用户只需执行下面的过程调用：

```
printList (Economics301ClassList)
```

就能得到预期结果。而且，如果以后我们想改变表的实际存储方式，那么就只需改变过程printList的内部操作，而对用户而言，仍然可以继续使用以前的同一个过程调用来完成打印操作。

```
procedure PrintList (List)
CurrentPointer←List的头指针
  while (CurrentPointer不是NIL) do
  (打印CurrentPointer指向的项中的名字；
  观察CurrentPointer指向的List项的指针单元中的值
  并重新将CurrentPointer赋值为那个值)
```

图8-19　一个打印链表的过程

问题与练习

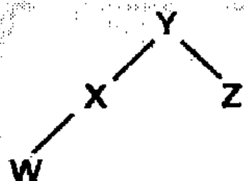
1. 请画出下面的数组是如何以行主序的方式在主存中存放的。

5	3	7
4	2	8
1	9	6

2. 如果一个二维数组是以列主序的方式，而不是行主序的方式存储的，那么请给出一个公式，用来查找该二维数组中的第i行第j列的元素。
3. 在C、C++、Java以及C#这些编程语言中，数组的下标是从0开始的，而不是从1开始。所以，数组Array

的第1行第4列的项可由`Array[0][3]`表示。这种情况下，翻译程序将使用什么样的地址多项式，把`Array[i][j]`这样的引用格式转化为存储器地址呢？

4. 什么条件表示链表为空？
5. 修改图8-19所示的过程，要求打印出某个指定的名字以后就停止打印。
6. 根据本节所提到的在一个连续的存储单元块中实现栈的技术，什么条件表示栈为空？
7. 请说明，在高级语言中，怎样用一维数组来实现一个栈？
8. 如果一个队列是运用本节所描述的循环方式实现的，那么当队列为空时，头指针和尾指针的关系如何？队列满时，关系又如何？怎么样来检测队列是满的还是空的？
9. 依据本节所讲的内容，当下面的一棵树采用的是左子指针和右子指针的方式存储时，试画出其在存储器中存放的情况。再者，利用本节所讲的树的另一种存储方式，再画出另一幅图，表示利用连续存储块存放时的情况。



8.4 一个简短案例

现在考虑一个按字母顺序排列名单的存储方案。假设对这个表进行如下操作：

查找（search）一个项，
按字母顺序打印（print）名单，
插入（insert）一个新项

其目标是开发一个存储系统以及实现这些操作，这样就实现了一个完整的抽象工具。

首先考虑存储这个表的几种可选的方法。如果按照链表方式来存储，就需要对表以串行的方式进行搜索，在第5章中就讨论过这样一个过程。如果表很长，那么这个处理过程的效率就非常低。所以要寻找另外一种实现方法，使得搜索过程能够利用到二分搜索算法（参见5.5节）。为了利用这种算法，必须能从所采用的存储系统中成功地找到这个表的较小部分里的中间项。其解决办法就是将表用二叉树的形式进行存储。也就是说，首先将表的中间项作为根节点，然后把表余下部分的头一半的中间项作为根节点的左子节点，把后一半的中间项作为根节点的右子节点。接下来，将表的剩余部分（除去根节点）的每四分之一部分的中间项再作为根节点的子节点的子节点，依次类推。例如，图8-20所示的树就表示了包含有A、B、C、D、E、F、G、H、I、J、K、L及M的一个字母表。（我们约定，当所讨论的表的一部分有偶数个项时，则取中间两项里的较大者为中间项。）

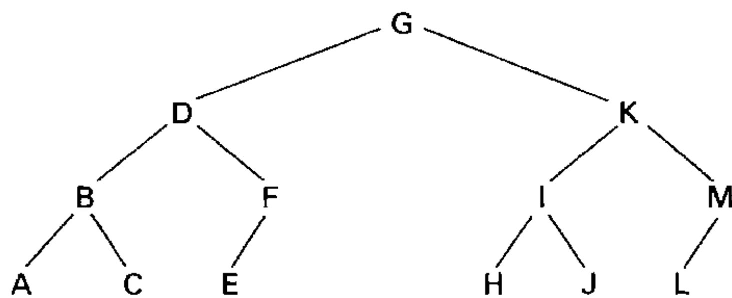


图8-20 字母A到M排列成一个有序树

为了搜索以这种方式存储的表，先将目标值与根节点的值进行比较，如果两者相等，则搜索成功。如果它们不相等，则依据目标值是小于或是大于根节点的值，分别转移至根的左子节

点或者右子节点。这样我们就可以发现，继续搜索的工作只需在表的一半中进行。这样的一种比较然后转移至子节点的过程一直继续下去，直至找到目标值（说明搜索工作成功了）或者最后遇到了NIL指针却还没有搜索到目标值（说明搜索工作失败了）为止。

图8-21表示的是，在链式的树结构情况下，如何来表示这样一种搜索过程。注意，图5-14所表示的就是二分搜索算法的原始描述，而这里的这个过程仅仅为该算法的一个细化而已，其区别从很大程度上讲是表面上的。原先所描述的算法是对表的小片段逐步进行搜索，而这里所描述的算法是对较小的子树进行逐步搜索（如图8-22所示）。

```
procedure Search (Tree, TargetValue)

if (Tree根指针=NIL)
 then
   (声明搜索失败)
 else
   (执行与下面的情况相关联的指令块)
  case 1:TargetValue=根节点的值
     (报告搜索成功)
  case 2:TargetValue<根节点的值
     (应用Search过程查看TargetValue是否在根的左子指针
    标识的子树中，并报告搜索的结果)
  case 3:TargetValue>根节点的值
(应用Search过程查看TargetValue是否在根的右子指针标识
的子树中，并报告搜索的结果)
  ) end if
```

图8-21 二分搜索用于作为链式二叉树实现的表

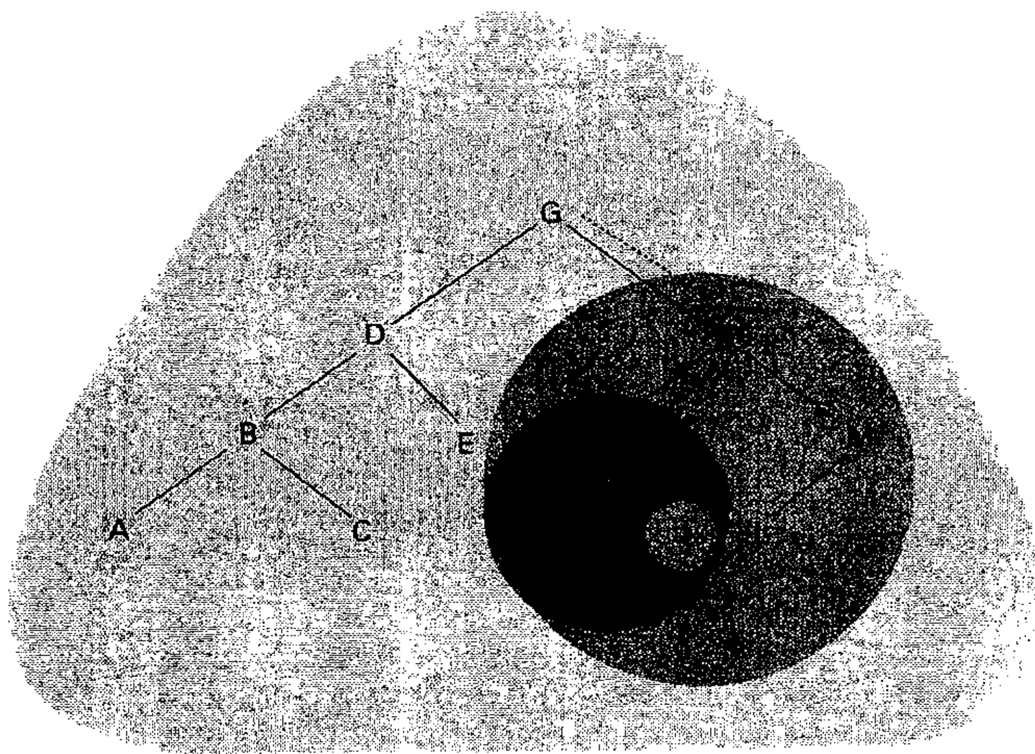


图8-22 利用图8-21中的过程搜索字母J所涉及的相继变小的树

你也许会这样认为：当把“表”存储为一个二叉树时，按照字母顺序对这个表进行打印的过程将会很困难。然而，为了能按照字母顺序打印出这个表，这里只需要先按字母顺序打印出左子树，然后打印出根节点，接下来再按字母打印出右子树即可（如图8-23所示）。因为，左子树所包含的元素都小于根节点的值，而右子树所包含的元素都大于根节点的值。到目前为止，该算法的逻辑框架如下表示：

```
if （树非空）
then （按照字母顺序打印左子树；
      打印根节点；
      按照字母顺序打印右子树）
```

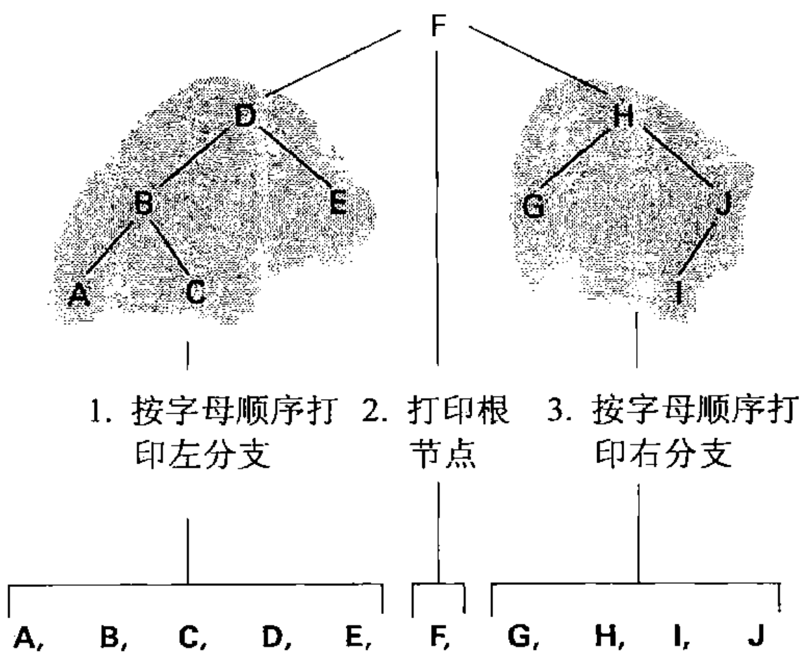


图8-23 按字母顺序打印出一个查找树

垃圾回收

随着动态数据结构的增大或缩小，存储空间也被占用或释放。回收不用的存储空间以备将来使用，这样一个过程称为**垃圾回收**（garbage collection）。许多场合都用到了垃圾回收机制。操作系统里的内存管理程序在分配和回收存储空间时必须执行垃圾回收工作。文件管理程序在计算机的大容量存储器上进行文件的存放和删除操作时，也要执行垃圾回收工作。此外，在分派程序控制下运行的任何进程，在给其分配的存储空间中也需要执行垃圾回收工作。

垃圾回收涉及一些难以捉摸的问题。对于链式结构，每当一个指向数据项的指针改变时，垃圾回收程序必须决定是否要回收指针原先指向的那个存储空间。在涉及有多路径指针交叉的数据结构中，这种问题就尤为复杂。不准确的垃圾回收例程会导致数据的丢失，或者是存储空间的利用率较低。例如，如果垃圾回收操作不能成功地回收存储空间，那么有效的存储空间就会越来越小，这种现象称为**内存泄露**（memory leak）。

这个框架中包含按照字母顺序打印左子树和右子树这两项任务，这两项任务本质上是原始打印任务的缩小版本。也就是说，打印一个树涉及打印子树的任务，这就使人想到运用递归方法来解决树的打印问题。

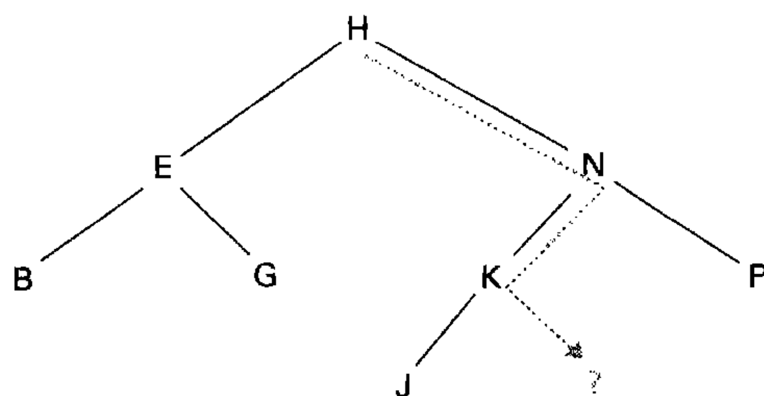
依据这条线索，可以把原先的设想扩展为打印二叉树的一个完整的伪代码过程，如图8-24所示。这里，将该例程命名为PrintTree，然后再调用PrintTree来打印左子树和右子树。注意，因为连续的递归过程中，每次递归所操作的树都要比启动递归的那个树要小，因而，递归过程的结束条件（遇到一个空子树）肯定会达到。

```
procedure PrintTree (Tree)

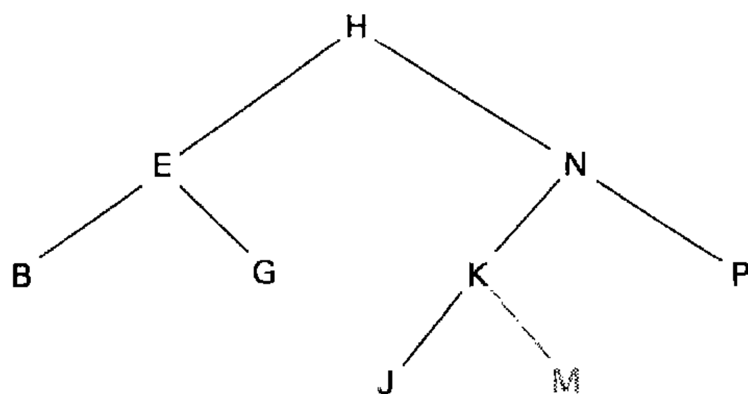
if (Tree非空)
  then (对以Tree中左分支出现的树应用PrintTree过程；打印
        Tree的根节点；对以Tree中右分支出现的树应用
        PrintTree过程)
```

图8-24 用于打印二叉树中数据的过程

在树中，插入一个新项的工作比起初看起来的也要容易。凭直觉也许会认为，插入新项只需先将树切开，为新项留出空间，但实际上，所添加的节点不论其值如何，只要作为一个新的叶子节点就可以将其插入到树中。为了给新项找到合适的位置，要沿着查找该项而遵循的那条路径往下走。由于该项并不在树中，所以我们将会查找到一个NIL指针。这个位置就是要存放新节点的合适位置（如图8-25所示）。事实上，找到的这个位置正是寻找新项所到达的地点。



(a) 为这个新项寻找一个空位置



(b) 这就是这个新项所应存放的位置

图8-25　把项M插入到其表项B、E、G、H、J、K、N、P以树结构存储的表中

对于链式树结构，这个处理过程的程序如图8-26所示。这里，首先对树进行搜索，找到要插入的值（称为NewValue），然后再把包含有NewValue的一个新叶子节点放到相应的位置。注意，如果在搜索过程中发现要插入的项已经在树中，则不必进行插入操作。

可以得出这样一个结论：包含了链式二叉树结构以及用于查找、打印、插入操作的这些过程的软件包提供了一个完整的包，该包可以作为我们假想应用的一个抽象工具。事实上，如果实现得恰当，可以使用这个软件包而无需关心底层的实际存储结构。利用软件包中的过程，用户可以想象出按字母顺序存放的名单表。而事实上，这个表的项都分散在不同的存储单元块中，并链接成一个二叉树。

问题与练习

1. 画一个二叉树，要求该树可以用来存放表R、S、T、U、V、W、X、Y和Z，以备将来搜索之用。
2. 简要说明，图8-21中的二分搜索算法在应用到图8-20中的树时，为查找项J所经历的路径。查找项P时的路径又是什么？
3. 画一个图，用来表示图8-24中打印树的递归算法用在图8-20所示的有序树中打印K节点时的活动状况。
4. 一个树结构，它的每个节点有26个子节点。试解释这样一个结构是如何对英语中的拼写正确的词汇进行编码的。

```
procedure insert (Tree,NewValue)

if (Tree的根指针=NIL)
  (设置根指针指向包含NewValue的新叶子节点)
  else (执行与相应情况对应的指令块)
    case 1:NewValue=根节点的值
        (什么也不做)
    case 2:NewValue<根节点的值
        (if根节点的左子指针=NIL)
            then (设置该指针指向包含NewValue的新叶
              子节点)
            else (应用Insert过程来插入NewValue到左子
              指针标识的子树中)
    case 3:NewValue>根节点的值
        (if根节点的右子指针=NIL)
            then (设置该指针指向包含NewValue的新叶
              子节点)
            else (应用Insert过程来插入NewValue到右子
              指针标识的子树中)
) end if
```

图8-26 在以二叉树存储的表中插入新项的过程

8.5 定制的数据类型

在第6章中，我们已经介绍过数据类型的概念，并讨论了如整型、实型、字符型及布尔型等基本数据类型。大多数编程语言都提供了这些基本数据类型。在本节中，我们讨论这样的一种方式，即程序员可以定义自己的数据类型，以便能更好地满足某个具体应用的需要。

8.5.1 用户自定义数据类型

如果除了编程语言中提供的那些基本数据类型外，还有其他数据类型可以用，那么对于表达一个算法来说，通常就比较简单了。基于这种原因，现代的许多编程语言都支持程序员利用基本数据类型作为构件块，来定义一些附加的数据类型。这些“自制”的数据类型的最基本的例子称为**用户自定义数据类型**（user-defined data types），其本质就是几个基本数据类型组合而成的具有同一名字的聚合体。

为了对此进行解释，这里假设要开发一个涉及许多变量的程序，而每个变量都有相同的异构数组结构，该结构中包含有名字、年龄以及技能级别。一种方法是将每个变量分别定义成异构数组（参见6.2节）。然而，更好的办法是将这个异构结构数组定义成一种新的（用户自定义的）数据类型，然后就把这种新的数据类型作为一种基本类型来使用。

为了实现上述想法，这里我们采用下列伪代码语句的形式来定义一个称为EmployeeType的新类型。

```
define type EmployeeType to be
{char  Name[25];
 int   Age;
 real  SkillRating;
}
```

这个异构结构中包含的组成元素有Name（字符型）、Age（整型）以及SkillRating（实型）。这样一来，就可以采用与基本数据类型相同的声明变量的方式，用这个新的数据类型来声明变量。也就是说，大多数编程语言都用语句

```
int x;
```

来声明变量x为整数。同样，变量Employee1也可以采用如下的语句来声明为EmployeeType类型：

```
EmployeeType Employee1;
```

于是，在以后的程序中，变量 Employee1 就将引用一整块存储单元，该存储块中包括员工的名字、年龄和技能级别。存储块中的各个项可以通过诸如 Employee1.Name 和 Employee1.Age 这样的方式来引用。所以，语句

```
Employee1.Age ← 26 ;
```

会被用来将值26赋给Employee1块中的Age元素。而且，语句

```
EmployeeType DistManager, SalesRep1, SalesRep2;
```

可以用来将 3 个变量 DistManager、SalesRep1 和 SalesRep2 声明为 EmployeeType 类型，正如如下形式的语句：

```
real Sleeve, Waist, Neck;
```

通常被用作将变量Sleeve、Waist、Neck声明为基本的real类型。

分清用户自定义数据类型与这个类型的一个实际项之间的区别非常重要。后者称为数据类型的一个**实例**（instance）。一个用户自定义的数据类型，其本质是一个用来构建数据类型实例的模板。该模板描述了这种类型的所有实例所具有的属性，但是它本身并非这种类型的一个实例（这就好比，饼干模是做饼干的模板，但其本身不是饼干）。在上面的例子中，用户自定义的数据类型EmployeeType被用来构建了该类型的三个实例，即DistManager、SalesRep1和SalesRep2。

8.5.2　抽象数据类型

尽管用户自定义数据类型的概念比较有用，但它还不足以创建完整意义上的新数据类型。一个完整的数据类型包含两部分：（1）一个预先确定的存储系统（如整型情况中的二进制补码系统和实型情况中的浮点系统），（2）一组预先定义的操作（如加和减）。具体来说，程序设计语言中的基本数据类型要与其基本的操作相联系。如果程序员将一个变量声明为一个基本类型，则无需进一步的定义，程序员就可以对该变量进行基本的操作。

然而，传统的用户自定义数据类型仅支持程序员定义新的存储系统，而没有提供对具有这些结构的数据进行处理的操作。为了对此进行说明，这里假设要在一个程序中创建和使用几个整数栈。其方法可以是，将每个栈实现成一个有20个整数值的同构数组。栈底中的项可以放在（压入）数组的第一个位置，栈的其他项将相继放在（压入）数组的高位项处（参见8.3节的问题与练习7）。再用一个整型变量作为栈指针，用来存放数组项的下标，而下一个栈项将会被压入到该数组中。因此，每个栈都由一个存放栈本身的同构数组和一个起着栈指针作用的整数组成。

为了实现这个构想，首先要用下列形式的语句来建立一个称为StackType的用户自定义类型：

```
define type StackType to be
{int StackEntries[20];
 int StackPointer = 0;
}
```

（注意，仿效如C、C++、C#及Java这些语言，就可以假设数组StackType的下标范围也是0~19，

这样StackPointer指针的初始值就为0。）做了这个声明之后，我们就可以通过如下语句来声明称为StackOne、StackTwo和StackThree的栈：

```
StackType StackOne, StackTwo, StackThree;
```

此时，变量StackOne、StackTwo和StackThree中的每一个都可以引用一个唯一的存储单元块，用以实现各自的栈。

但是，如果现在要把25这个值压入到StackOne，那么该怎么办？当然，我们希望能屏蔽掉以栈的实现为基础的数组结构的细节，而仅仅将栈作为一种抽象工具来使用，即可能会用类似于

```
push(25, StackOne)
```

这样的一个过程调用。但是，如果不定义称为push的相应过程，那么这样的一条语句就不可用。要完成对StackType类型的变量的操作还包括从栈中弹出项、检查栈是否为空以及检查栈是否已满，而这所有的操作都要求再定义相应的过程。简而言之，我们定义的StackType数据类型并不包括与之关联的所有特性。

对此问题的解决办法是，对定义语句define type进行扩展，使其既包括数据的描述，又包括相关的处理过程。例如，可以这样写：

```
define type StackType to be
{int StackEntries[20];
 int StackPointer = 0;
 procedure push(value)
   {StackEntries[StackPointer] ← value;
    StackPointer ← StackPointer + 1;
   }
 procedure pop . . .
}
```

这些语句是用来表明：StackType类型与称为StackEntries和StackPointer的变量相关，也与称为push和pop的过程相关。（为了简化，这里包括了一个非常简单的push过程版本。实际上，这个过程在插入一个新项前，应该保证栈不能满。）

利用这个扩展过的StackType类型的定义，就可以通过语句：

```
StackType StackOne, StackTwo, StackThree;
```

将StackOne、StackTwo和StackThree声明为栈。然后，通过诸如下面的语句：

```
StackOne.push(25);
```

把项压入这些栈中。该语句表示用值25作为实际参数，进而执行与StackOne相关联的push过程。

这里，将包含了操作定义的用户自定义数据类型称为**抽象数据类型**（abstract data type）。所以，相对于那些更为基本的用户自定义数据类型而言，抽象数据类型就是完整的数据类型。在20世纪80年代，抽象数据类型出现在Ada等语言中，这就代表了在编程语言设计方面前进了一大步。今天，面向对象语言提供了称为类的抽象数据类型的扩展版本，在下一节中将会介绍到。

问题与练习

1. 数据类型与该数据类型的一个实例之间有什么不同？
2. 用户自定义数据类型与抽象数据类型之间有什么不同？
3. 请描述用来实现表的抽象数据类型。
4. 请描述用来实现支票账户的抽象数据类型。

*8.6 类和对象

在第6章中我们已经讨论过，面向对象范型导致了系统可由称为对象的单元组成，而任务是通过对象之间的交互来完成的。每个对象就是一个实体，并响应来自其他对象的消息。对象由称为类的模板来描述。

在许多方面，这些类实际上就是抽象数据类型（它们的实例称为对象）的描述。事实上，在许多流行的面向对象的程序设计语言中，用来定义类的语句与上节中介绍的define type语句非常相似。举例来说，图8-27展示了，在Java语言和C#语言中，如何定义StackOfIntegers类。（在C++语言中，等价类的定义具有相同的结构，但在语法上稍微有所不同。）可以注意到，这里的类与上一节描述抽象数据类型StackType所用的define type语句之间有些类似。这里所描述的类/类型包括了一个称为StackEntries的整型数组，一个用来确定数组中栈顶位置的整数StackPointer，以及一些用来处理栈的过程。

```
class StackOfIntegers
{private int[] StackEntries = new int[20];
 private int StackPointer = 0;


 public void push(int NewEntry)
 {if (StackPointer < 20)
    StackEntries[StackPointer++] = NewEntry;
 }

 public int pop()
 {if (StackPointer > 0) return StackEntries[--StackPointer];
  else return 0;
 }
}
```

图8-27　Java和C#语言中实现的整数栈

在Java或C#程序中，可以利用这个类作为模板，用以下语句来创建一个名为StackOne的对象：

```
StackOfIntegers StackOne = new StackOfIntegers();
```

或者在C++程序中，用以下语句来创建该对象：

```
StackOfIntegers StackOne();
```

在以后的程序中，使用以下语句，可以将值106压入到StackOne栈中：

```
StackOne.push(106);
```

或者可以用下面的语句把StackOne的栈顶元素读取到变量OldValue中：

```
OldValue = StackOne.pop();
```

这些特征与抽象数据类型的那些特征本质上是一样的。然而，类与抽象数据类型之间还是有些区别的。前者是后者的扩展。例如，如在6.5节中已经介绍过的，面向对象语言允许类从其他的类继承属性，并包括称为构造函数的特殊方法，当创建对象时，用其来定制个性化的对象。而且，类通常都有不同程度的封装性（参见6.5节），这样就可以避免其实例的内部属性受非正常的访问。最后，类可以作为一种对相关过程进行分组的方法，因此，类可以只由过程定义组成。从这个意义上讲，可以把类称为抽象类型，而不是抽象数据类型。

标准模板库

本章所讨论的数据结构已经成为标准的编程结构，事实上，因为其标准性，以至于许多编程环境都把它们当作原语一样对待。在C++编程环境中就可以找到一个例子，即通过标准模板库（Standard Template Library，STL）使该环境的功能更为强大。STL中包含了一组预先定义好的类，这些类是用来描述常用的数据结构。因此，通过在C++程序中并入STL的这种方式，程序员就可以从描述这些结构细节的工作中解放出来；他们只需声明所指的标识符是什么类型就行，就像在8.6节中将`StackOne`声明为`StackOfIntegers`类型那样。

最后可以得出结论：类和对象的概念体现了程序中数据抽象的表示技术又前进了一大步。事实上，正是由于这种以方便的方式来定义和使用抽象的能力，才导致了面向对象设计范型的流行。

问题与练习

1. 抽象数据类型与类在哪些方面类似？在哪些方面存在着不同？
2. 类与对象有什么不同？
3. 请描述一个类，要求用该类作为构建整数队列类型对象的模板。

*8.7 机器语言中的指针

本章已经介绍过指针，并介绍了如何利用指针来构建数据结构。本节我们将讨论如何在机器语言中处理指针。

假设我们要用附录C中所描述的机器语言写一个程序，要求从图8-12所示的栈中弹出一个项，然后将其放入到一个通用寄存器中。换句话说，就是要将存储单元中的内容加载到一个寄存器中，而这个存储单元包含的是栈顶的项。我们的机器语言提供了两条指令用于加载寄存器：一条指令是用操作码2，另一条指令是用操作码1。回想一下，在操作码2的情况中，操作数字段包含了被加载的数据，而在操作码1的情况中，操作数字段则包含了被加载数据的地址。

由于不知道内容是什么，所以用操作码2达不到预期目的。而且，不知道地址，也不能用操作码1。毕竟，在程序执行的时候，栈顶的地址会发生变化。然而，我们知道了栈指针的地址。也就是说，知道了所要加载数据的地址的位置。于是，我们需要的就是第3个用于加载寄存器的操作码，在这条指令中，操作数字段包含了指向被加载的数据指针的地址。

为了实现这个目标，我们对附录C中的机器语言进行扩展，使其包含操作码D。使用这个操作码的指令可能具有这样的形式，即DRXY，这就表示将地址为XY的存储单元的内容加载到寄存器R中（如图8-28所示）。所以，如果栈指针在地址AA的存储单元中，则指令D5AA就实现了将栈顶的数据加载进寄存器5中。

然而，这条指令并没有完成出栈操作。我们还必须将栈指针减1，以便让它指向新的栈顶。这也就是说，在加载指令之后，机器语言程序还必须将栈指针加载到一个寄存器，将其减去1，然后再把结果存回到存储器。

如果不用存储单元，而用某个寄存器来作为栈指针，那么就可以减少栈指针在寄存器与存储器间的来回移动。但是，这也就意味着必须重新设计加载指令，这样它就要求指针在寄存器中，而不是在主存中。这样一来，我们不使用早些时候那个方案，而是用操作码D定义一条指令，令其具有DR0S的形式，这就表示将寄存器S所指的存储单元的内容加载到寄存器R（如图8-29所示）。于是，一个完整的出栈操作就可以这样来完成：在这条指令之后添加一条指令（或

几条指令），将存放在寄存器S中的值减去1。

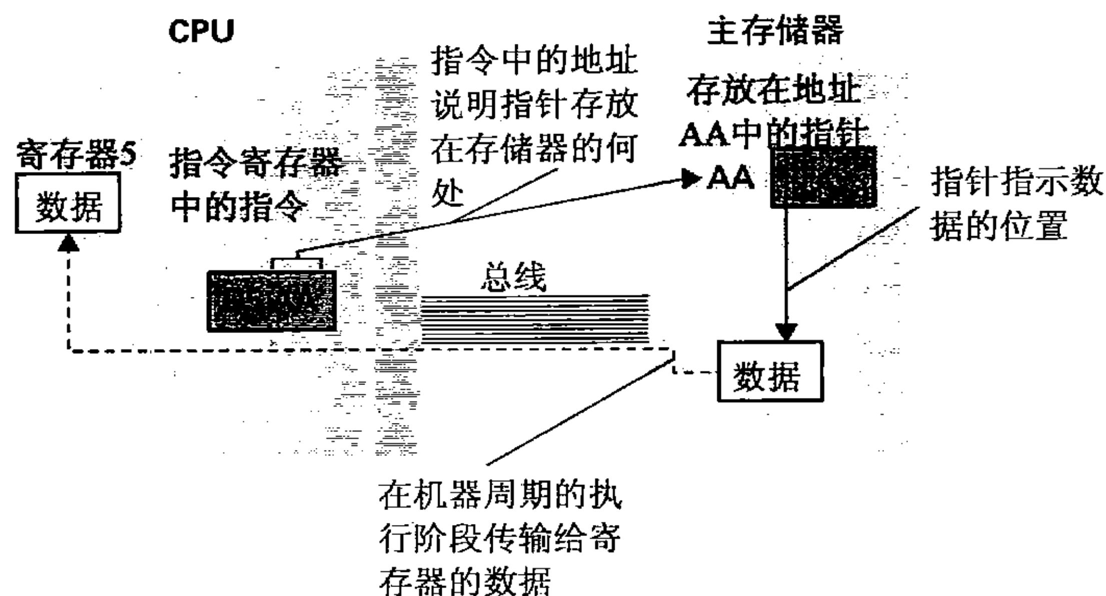


图8-28　利用指针扩展附录C中机器语言的首次尝试

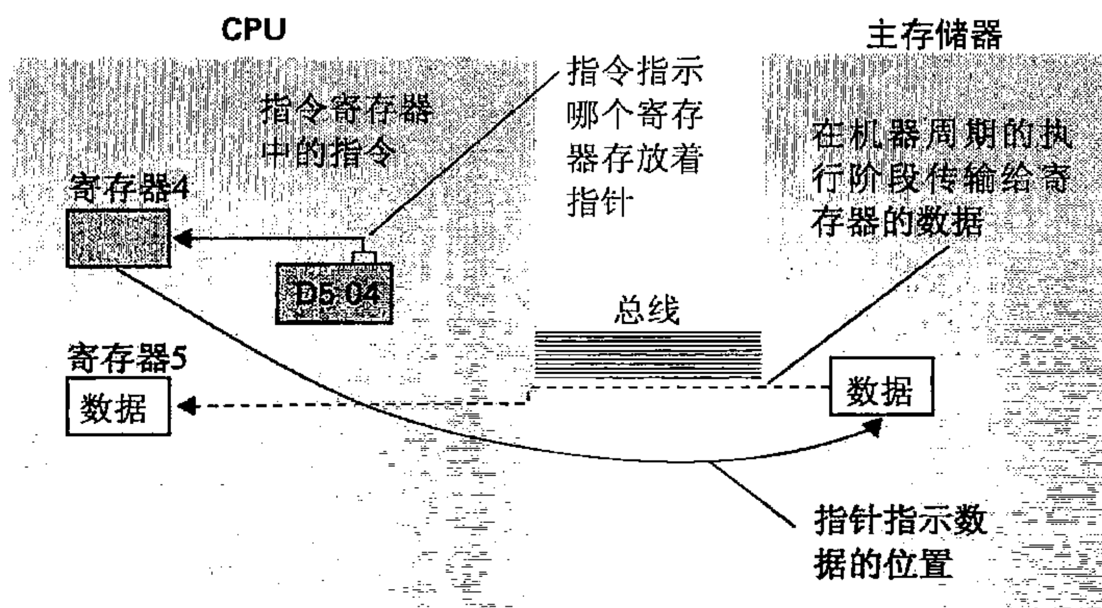


图8-29　把存放在寄存器中的一个指针指向的存储器单元的内容加载到一个寄存器中

注意，要实现入栈操作，还需要一条类似的指令。所以，还需要对附录C中所描述的机器语言做进一步的扩展，使其引入操作码E，这样一来，ER0S的指令形式就表示把寄存器R的内容存放到由寄存器S所指向的存储单元中。同样，为了完成入栈操作，在这条指令之后添加一条指令（或几条指令），将寄存器S中的值加上1。

我们所提出的这些新的操作码D和E，不仅说明了所设计的机器语言是如何处理指针的，它们还说明了最初的机器语言中没有提到的寻址技术。正如附录C中所提到的，机器语言用两种方式来确定一条指令中所涉及的数据。第一种方式就是通过一条操作码为2的指令来表示。在这里，操作数字段就明确包括了所涉及的数据。这种寻址技术称为**立即寻址**（immediate addressing）。确定数据的第二种方式则用操作码为1和3的指令来表示。在这里，操作数字段包含的是所涉及的数据的地址。这种寻址技术称为**直接寻址**（direct addressing）。然而，我们所提出的新操作码D和E则表明还有另一种确定数据的形式。这些指令的操作数字段包含的是数据地址的地址。这种寻址技术称之为**间接寻址**（indirect addressing）。所有的这3种寻址技术在今天的机器语言中是比较常见的。

问题与练习

1. 假设附录C中所描述的机器语言按照本节最后的建议进行了扩展，而且，假定寄存器8中的内容为模式DB，而地址为DB的存储单元中的内容为模式CA，并且地址为CA的存储单元的内容为模式A5。请

问：在执行了下面的每条指令后，寄存器5中的位模式是什么？

a. 25A5　　b. 15CA　　c. D508

2. 利用本节最后所描述的扩展，写一段完整的机器语言例程，来完成出栈操作。假设栈是按图8-12所示的方式实现的，栈指针在寄存器F中，并且，栈顶出栈后压入寄存器5中。
3. 利用本节最后所描述的扩展，写一段程序，将从地址A0开始的5个连续的存储单元的内容复制到从地址B0开始的5个存储单元。这里假设程序的起始地址为00。
4. 在本章中，已经介绍过DR0S这样一种形式的机器指令。假设将这个形式扩展为DRXS，其意义为：把寄存器S中的值加上值X，然后将结果所指向的数据加载到寄存器R中。这样一来，通过读取寄存器S中的值，再加上值X，就可以得到指向数据的指针。寄存器S中的值不会发生变化。（如果寄存器F的内容为04，那么指令DE2F就把地址为06的存储单元的内容加载到寄存器E中，而寄存器F的值保持04不变。）请问：这条指令的优点是什么？如果一条指令的形式为DRTS，即表示"把寄存器S的值和寄存器T的值相加，然后把所得的结果所指向的数据加载到寄存器R中"，那么这条指令又有什么优点？

复习题

（带*的题目涉及选读章节的内容。）

1. 当下列数组分别以行主序和列主序在机器的存储器中存放时，试画图说明该数组的存放情况。

A	B	C	D
E	F	G	H
I	J	K	L

2. 假设一个6行8列的同构数组是按行主序存放的，其起始地址为20（十进制）。如果数组中的每个项只需要一个存储单元，数组中的第3行第4列的项的地址是多少？如果每个项需要两个存储单元，那么结果又如何？
3. 假设第2题中的数组是以列主序而不是以行主序存放的，请重新做第2题。
4. 如果想利用传统的一维同构数组来实现动态表，那么会带来怎样的复杂问题？
5. 请描述一种用来存放三维同构数组的方法。请问这里用来定位第i面、第j行、第k列的项的地址多项式是什么？
6. 假设字母表A、B、C、D、E、F和G存放在一个连续的存储单元块中，假设要保持表的字母顺序，插入字母D则需要进行哪些操作？
7. 下表表示的是计算机主存中的一些存储单元的内容以及每个单元的地址。注意，其中有些单元包含字母表中的字母，而每个这样的单元后面都跟随一个空单元。在这些空单元中填入适当的地址，使得每个包含字母的单元及其后的单元一起，构成一个链表中的项，并且该链表要按字母顺序排列。（这里用0来表示NIL指针。）这里的头指针包含的内容是什么？

地　址	内　容
11	C
12	
13	G
14	
15	E
16	
17	B
18	
19	U
20	
21	F
22	

8. 下表代表的是计算机主存中一个链表的一部分。表中的每项由两个单元组成：第一个单元包含的是字母表中的字母，第二个单元包含的是指向链表下一项的指针。请改变指针，以便字母N不再出现在表中；然后，用字母G代替字母N，并改变相应的指针，使新字母按字母顺序出现在表中的合适位置。

地　址	内　容
30	J
31	38

（续）

地址	内容
32	B
33	30
34	X
35	46
36	N
37	40
38	K
39	36
40	P
41	34

9. 下面的表所表示的链表与前面几题所使用的格式相同。如果头指针包含的值是44，那么这个表所表示的名字是什么？改变指针，使得这个表包含名字Jean。

地址	内容
40	N
41	46
42	I
43	40
44	J
45	50
46	E
47	00
48	M
49	42
50	A
51	40

10. 下面的哪一个例程能够正确地做到：将NewEntry项直接插入到链表中名为PreviousEntry的项的后面？另外一个例程的错误在什么地方？

例程1：

1. 将PreviousEntry指针字段的值复制到NewEntry的指针字段。
2. 将PreviousEntry指针字段的值改成NewEntry的地址。

例程2：

1. 将PreviousEntry指针字段的值改成NewEntry的地址。
2. 将PreviousEntry指针字段的值复制到NewEntry的指针字段。

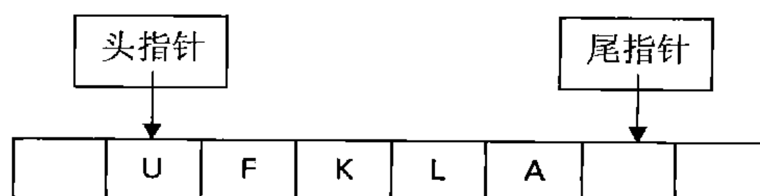
11. 设计一个过程，将两个链表连接起来（也就是说，把一个链表放到另一链表的前面，形成一个链表）。
12. 设计一个过程，将两个排序过的邻接表合并成一个排序过的邻接表。如果表是链式的，那么结果又如何？
13. 设计一个过程，对一个链表进行反向排列。
14. a.设计一个算法，利用栈作为辅助存储结构，以反序打印出一个链表。
    b.设计一个递归过程来完成同样的任务，而不显式使用栈结构。这个递归的解决方案所涉及的栈会采用什么形式？
15. 有时候，一个单链表可以有两种不同的顺序，只要为每一项附加两个指针，而不是一个指针即可。请填充下表，使得通过紧跟每个字母的第一个指针，就可以找到名字Carol；而通过紧跟每个字母的第二个指针，就可以按照字母顺序找到字母。所表示的这两个表的头指针分别包含什么值？

地址	内容
60	O
61	
62	
63	C
64	
65	
66	A
67	
68	
69	L
70	
71	
72	R
73	
74	

16. 下表表示的是文中所讨论过的，存放在连续存储单元块中的一个栈。如果这个栈的栈底地址是10，而栈指针包含值12，那么，一个出栈指令取出的是什么值？执行出栈操作后，栈指针中的值是什么？

地址	内容
10	F
11	C
12	A
13	B
14	E

17. 在第16题中，如果执行的指令是向栈中压入字母D，而不是弹出一个字母，那么请画出一个表来显示存储单元中最后的内容。在执行入栈指令后，栈指针中的值是什么？
18. 设计一个过程，从一个栈中删除栈底项，而栈中的其他项保持不动。这里只能用出栈和入栈操作来访问栈。为了解决这个问题，还需要用到什么样的辅助存储结构？
19. 设计一个过程，比较两个栈的内容。
20. 假设给你两个栈，如果一次只允许你从一个栈移动一个项到另一个栈，那么原始的数据将可能进行怎样的一些重排？如果给你3个栈，那么会有怎样的安排？
21. 假动设给你3个栈，并且一次只允许你从一个栈移动一个项到另一个栈。设计一个算法，把其中一个栈中的两个相邻项颠倒。
22. 假设要创建一个存放名字的栈，其中名字的长度不同。如果把名字存放在分散的存储区域，再建立一个指向这些名字的指针的栈，而不是在栈中存放名字本身，请解释一下为什么这样做更方便？
23. 队列在存储器中是向其头的方向移动，还是向其尾的方向移动？
24. 假设要实现一个“队列”，而该队列中的新项都有相应的优先级。这样，一个新的项就会被放在那些优先级较低的项前面。请描述一个实现这种“队列”的存储系统，并证明你的结论的正确性。
25. 假设队列中的每个项需要一个存储单元，其头指针包含值11，尾指针包含值17。那么请问，当向队列中插入了一项，同时移走两项，那么这些指针的值又为多少？
26. a. 假设一个队列是以循环队列的形式实现的，其状态如下图所示。请画出一个图，用来表示处在插入字母G和R，移走3项，再插入字母D和P之后的结构。



    b. 在(a)中，如果在没有移出任何字母之前，就插入字母G、R、D和P，那么会发生什么样的错误？
27. 在用高级语言编写的一个程序中，请描述一下怎样用数组来实现队列。
28. 假设有两个队列，一次只允许从一个队列的头部移一个项到任何一个队列的队尾。请设计一个算法，把其中一个队列的相邻两项颠倒。
29. 下表表示的是存放在计算机存储器中的一棵树。树的每个节点有3个单元。第一个单元存放的是数据（字母），第二个单元包含的是指向该节点左子节点的指针，第三个单元包含的是指向该节点右子节点的指针。0值代表NIL指针。如果根指针的值是55，那么请画出这棵树。

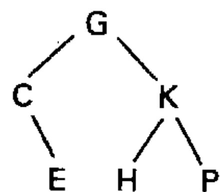
地　址	内　容
40	G
41	0
42	0
43	X
44	0
45	0
46	J
47	49
48	0
49	M
50	0
51	0
52	F
53	43
54	40
55	W
56	46
57	52

30. 下表表示的是计算机主存中一个单元块的内容。注意，一些单元中包含的是字母表中的字母，而每个这样的单元后面都跟有两个空单元。填充这些空单元，使得这个存储块表示下面的那棵树。这里用字母后的第一单元作为指向左子节点的指针，而接下来的那个单元则作为指向右子节点的指针。用0表示NIL指针。根指针的值应该为多少？

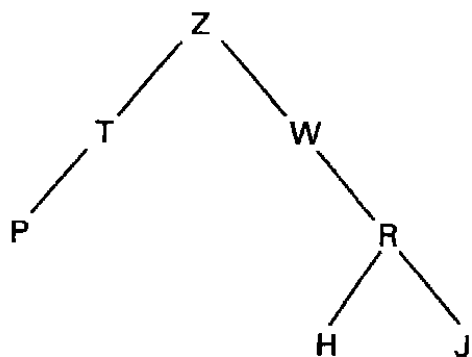
地　址	内　容
30	C
31	
32	
33	H
34	
35	
36	K

（续）

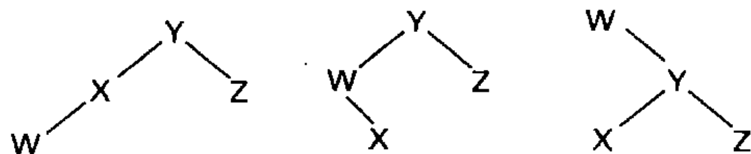
地 址	内 容
37	
38	
39	E
40	
41	
42	G
43	
44	
45	P
46	
47	



31. 设计一个非递归算法来代替图8-21所示的递归算法。
32. 设计一个非递归算法来代替图8-24所示的递归算法。利用一个栈来控制必要时的回溯。
33. 应用图8-24中所示的打印树的递归算法。画出一个图，用来表示在打印X节点时该算法的嵌套活动（以及每个活动的当前位置）。
34. 在保持根节点相同，而且也不改变数据元素的物理位置的情况下，请改变第29题中树的指针，使得图8-24中所示的树的打印算法按字母顺序打印出节点。
35. 如果下面的二叉树不用指针，而以8.3节中所描述的用连续存储单元块存放，请画出一个图，用来表示该二叉树在存储器中是如何存储的。



36. 假设如8.3节中所描述的，用连续存储单元来表示二叉树，其值分别为A、B、C、D、E和F。请画出这个树的图。
37. 举一个例子，可以把一个表（概念结构）实现为一棵树（实际的底层结构）。再举一个例子，可以把一棵树（概念结构）实现为一个表（实际的底层结构）。
38. 文中所讨论的链式树结构包含有指针，这就使得可以沿着树从父节点下移到子节点。请描述一个指针系统，可以沿着树从子节点上移到父节点。兄弟节点之间的移动又如何？
39. 请描述一个数据结构，要求该数据结构适用于表示下棋游戏时的棋盘布局。
40. 如果按照图8-24所示的算法，判断下列几棵树，哪棵树的节点会按照字母顺序打印出来？



41. 修改图8-24中的过程，使得能按倒序打印出“表”。
42. 描述一个树结构，将其用来存放一个家族的家谱。对该树会进行一些什么样的操作？如果该树是用链式结构来实现的，每个节点应该关联一些什么样的指针？假设这个树就是按照你刚才所描述的指针，并以链式结构实现，请设计相应的过程来完成你所定义的上述操作。利用你设计的存储系统，解释一下如何才能找到一个人的所有兄弟。
43. 如果一棵树是按图8-20所示的形式存放的，请设计一个过程来从这棵树中找到并删除给定的值。
44. 在树的传统实现方式中，构造的每个节点都会为其每个可能的子节点分别留有指针。所设计的这种指针的数目决定了任何节点所拥有的子节点的最大数目。如果一个节点的子节点数目少于指针数目，则将有些指针简单地置为NIL即可。但是，这样的一个节点不可能拥有比指针数目更多的子节点。请描述一下，如何实现一棵树，而不限制其节点所拥有的子节点数。
45. 利用8.5节介绍的difine type伪代码语句，定义一个用户自定义数据类型，用来表示关于公司员工情况的数据（如名字、地址、工作职务、工资级别等）。
46. 利用8.5节介绍的difine type伪代码语句，拟定一个表示名单的抽象数据类型。具体来说，用什么样的结构来包含这个名单？提供什么样的过程来处理这个名单？（没有必要包括过程的详细说明。）

47. 利用8.5节介绍的difine type伪代码语句，拟定一个表示队列的抽象数据类型。然后再给出伪代码语句，说明如何创建这个类型的实例，以及如何在这些实例中插入和删除项。
48. a.抽象数据类型和基本数据类型之间的区别是什么？
    b.抽象数据类型与用户自定义数据类型之间的区别是什么？
49. 试确定用来表示地址簿的抽象数据类型中可能会出现的数据结构和过程。
50. 试确定用来表示视频游戏中一个简单航天器的抽象数据类型中可能出现的数据结构和过程。
*51. 修改图8-27，使得该类定义的是一个队列，而不是栈。
*52. 类与传统的抽象数据类型相比，在哪个方面更通用？
*53. 利用8.7节最后描述的DR0S和ER0S的指令形式，写一个完整的机器语言例程，向图8-12中所实现的栈中压入一个项。这里假设栈指针放在寄存器F中，而要压入的项在寄存器5中。
*54. 假设链表中的每个项都由一个存放数据的存储单元以及指向下一项的指针构成。而且，假设一个位于存储器地址A0处的新项要插入到位于地址B5处和C4处的项之间。利用附录C中描述的语言，以及8.7节最后所描述的附加操作码D和E，写一个机器语言例程来实现这个插入操作。
*55. 8.7节所描述的DR0S指令形式与DRXY指令形式相比有什么优势？在8.7节的问题与练习4中所描述的DRXS指令形式与DR0S指令形式相比有什么优势？

社会问题

下面的问题有助于你分析一些与计算领域相关的道德、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 假设一个软件分析师设计了一个数据组织方式，能在一个特定的应用中有效地处理数据。那么该如何保护对这个数据结构的权益呢？数据结构是否是一种思想表达（好比一首诗），所以也可以通过版权来进行保护？还是也像算法一样，钻了同样的法律空子？用专利法呢？
2. 在何种的程度上，错误的数据比没有数据更糟糕？
3. 在许多应用程序中，栈可以扩展到多大取决于可用的存储器容量有多大。如果可用的空间已经耗尽，那么所设计的软件就会产生一条消息，诸如“栈溢出”等，并终止程序。在大多数场合，这种错误从不会发生，而且用户也从不会意识到这个错误。但是，如果这种错误发生了，并且丢失了敏感的数据，那么谁将对此负责？软件的开发者怎样减轻自己的责任？
4. 在基于指针系统的数据结构中，删除一个项通常是通过改变指针，而不是擦掉存储单元来办到的。这样一来，当链表的一项被删除后，这个删除的项实际上还留在存储器中，直到这个存储空间被其他的数据用掉。这种被删除的数据存留的状况会产生什么样的道德和安全方面的问题？
5. 数据和程序可以方便地从一台计算机传送到另一台计算机。这样一来，一台机器所拥有的知识也可以很容易地传送给许多机器。相反，对人而言，要把知识从一个人传给另一个人，有时候得花很长的时间。例如，一个人要教会另一个人一种新语言，那得花时间。如果机器的能力开始挑战人的能力，那么这种在知识传输率上的反差将意味着什么？
6. 利用指针可以将相关的数据在计算机存储器中链接起来，其链接方式使人联想到，信息在人脑中也是采用这种方式关联起来的。那么，这样一种在计算机存储器中的链接与人脑中的链接有怎样的相似之处？它们的不同点是什么？如果尝试着把计算机建造得与人脑更相像，那么这在伦理上是否可取？

7. 计算机技术的普及是否已经产生了新的道德问题，或者只是提供了一个新的环境，而在这样的环境之中，原来的那些道德规范理论是否还适用？
8. 假设计算机科学导论教材的作者想用一些程序的例子来说明文中的概念。为了简明，许多例子必须对实际的专业优质软件做些简化。作者知道，这些例子会被不持怀疑态度的读者所使用，并最终用到一些重要的软件应用中去，而这些应用更适合采用较为健壮的技术。作者应当采用这些简化版的例子，即使因为简化降低了它们的价值，仍坚持认为所有的例子是健壮的，还是应当拒绝使用这些例子，除非它们在简明性和健壮性方面都能得到保证？

课外阅读

Carrano, F. M. *Data Abstraction and Problem Solving with C++: Walls and Mirrors*, 5th ed. Boston, MA: Addison-Wesley, 2007.

Carrano, F. M. and J. Prichard.*Data Abstraction and Problem Solving with Java: Walls and Mirrors*, 2nd ed. Boston, MA: Addison-Wesley, 2006.

Gray, S. *Data Structures in Java: From Abstract Data Types to the Java Collections Framework*. Boston, MA: Addsion-Wesley, 2007.

Main, M. *Data structures and Other Objects Using Java*, 3rd ed. Boston, MA: Addison-Wesley, 2006.

Main, M. and W. Savitch. *Data structures and Other Objects Using C++*, 4th ed. Boston, MA: Addison-Wesley, 2010.

Shaffer, C. A. *Practical Introduction to Data Structures and Algorithm Analysis*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2001.

Weiss, M. A. *Data Structures and Problem Solving Using Java*, 3rd ed. Boston, MA: Addison-Wesley, 2006.

Weiss, M. A. *Data Structures and Algorithm Analysis in C++*, 3rd ed. Boston, MA: Addison-Wesley, 2007.

Weiss, M. A. *Data Structures and Algorithm Analysis in Java*, 2nd ed. Boston, MA: Addison-Wesley, 2007.

第9章

数据库系统

数据库是这样一个系统，它将庞大的数据集合转化成一个抽象工具，为用户提供一个简便的方式查找并提取相关的信息项。本章将讨论这个主题，另外还将讨论一个数据挖掘相关领域的主题。数据挖掘技术，即一种从庞大的数据集合和传统文件结构中发现隐藏模式的技术，能够为今天的数据库和数据挖掘系统提供许多基本的工具。

本章内容

当今的技术已经能够存储相当大量的数据，但是，如果我们不能提取与手头工作相关的有用信息项，那么这样的数据集合就是无用的。本章将研究数据库系统，并弄清这些系统是怎样利用抽象工具从庞大的数据集合中提取出有用的信息。作为相关主题，还要研究数据挖掘，即一个与数据库技术密切相关的快速发展的领域，其目标是开发一种技术，用于在数据集中确定和寻找数据的模式。此外，我们还将学习传统文件结构的原理，因为它支撑了现在的数据库和数据挖掘系统。

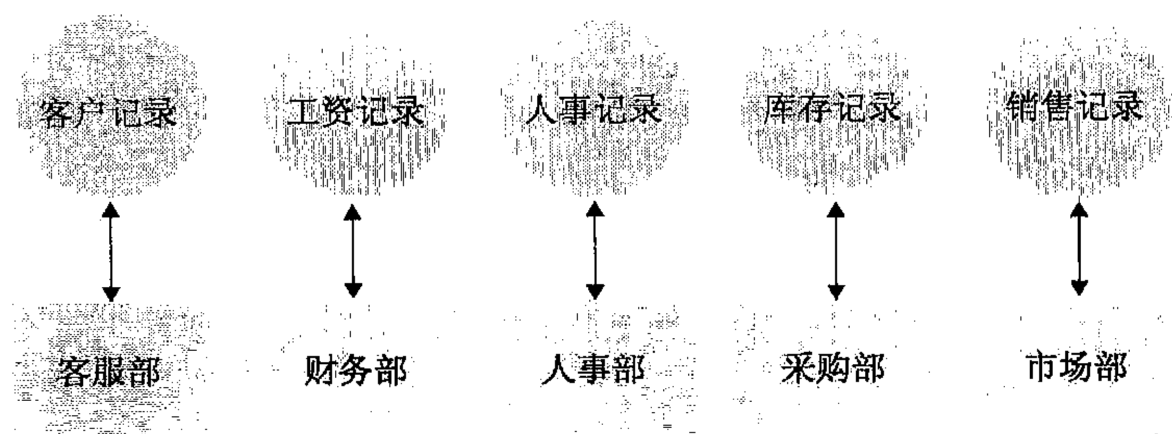
9.1 数据库基础

数据库（database）是指一种多维的数据集合。之所以说是多维的，是因为在这种集合中，通过数据项间的内部链接，信息可以从不同的角度来获取。这与传统的文件系统（参见9.5节）不同，传统的文件系统，有时也称为**平面文件**（flat file），是一种一维的存储系统，因为它只从一个角度来展示信息。比如，一个包含作曲家及其作品信息的平面文件，也许只能提供一个按作曲家分类的作品清单；而对于一个数据库来说，它可以呈现某一作曲家的所有作品，也可以是某一类音乐作品的所有作曲家，还可以是改写了其他作曲家作品的那些作曲家。

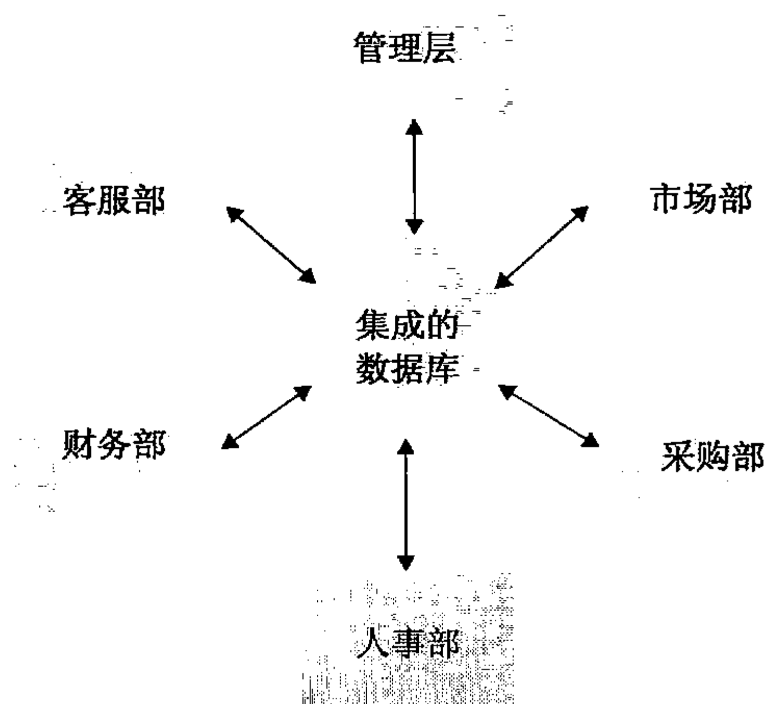
9.1.1 数据库系统的重要性

从历史发展角度来看，计算机广泛应用到信息管理领域时，每个应用都是作为独立系统来实现的，各有一套自己的数据。工资用工资单文件处理，人事部门有自己的职工记录，库存通过库存文件来管理。这就意味着，许多只是一个部门需要的信息在整个公司里会被复制，而许多虽然不同但相互关联的数据项却又存放在不同的系统中。在这种背景下，数据库系统应运而生，它作

为一种信息集成的手段，通过特定的组织来存放和维护数据（如图9-1所示）。利用这样一个系统，可以根据相同的销售数据来确定进货订单，生成市场变化趋势的报告，指导广告发放，并向最可能给予此种产品好评的客户提供相应的产品发布信息，使得销售团队取得更好的业绩。



(a) 面向文件的信息系统



(b) 面向数据库的信息系统

图9-1 文件与数据库结构的比较

这样的信息集成池提供了有价值的资源，只要能通过有意义的方式来获取这些有效信息，就可以通过它做出管理决策。因此，数据库的研究重点在于开发一些技术，把数据库中的信息提供给决策过程。在此方面人们已经取得了很大进展。如今，数据库技术与数据挖掘技术相结合，已成为一种重要的管理工具，使组织的管理层从涵盖组织和其环境的各个方面的大量数据中提取出相应的信息。

而且，数据库系统已经成为支撑万维网中许多流行网站的基础技术。如Google、eBey和Amazon等站点的基础主题是提供客户端与数据库间的接口。为了响应客户端的请求，服务器查询数据库，以网页的形式组织查询结果，并把网页发送给客户端。这样的Web接口已经使数据库技术承担了一个新角色，数据库不再是存储公司记录的一种手段，而成为了公司的产品。实际上，通过结合数据库技术和Web接口，因特网已经成为主要的全球信息源。

9.1.2 模式的作用

数据库技术的迅速发展有一个缺点，即潜在的敏感数据被未经授权的人访问。某人在公司网站上下了一份订单，但他不应该有权限访问公司的财务数据；类似地，负责公司福利的部门的员工有权访问公司员工记录，但不应该有权访问公司的库存或销售记录。因此，对数据库中

信息的访问控制能力与共享它的能力同等重要。

为了让不同的用户访问数据库中不同的信息，通常数据库系统都依赖模式和子模式。**模式**（schema）是整个数据库结构的一个描述，数据库软件用它来维护数据库。**子模式**（subschema）只是与特定用户需求相关的那部分数据库的一个描述。例如，一个大学数据库的模式应当说明，每个学生记录包含的条目除了学习成绩外，还有现阶段的联系地址、电话，还要说明每个学生的记录要与其指导教师的记录相链接。同样，每个教师的记录要包含个人地址、工作经历等。基于这样一个模式，要维持一个链接系统，最终使得学生的信息与教师的工作经历相关联。

为了使大学的注册会员不能利用这种链接关系来访问教师的专有信息，就必须限制注册会员只能访问数据库的子模式。这种子模式中，对教师记录的描述并不包括其工作经历。注册会员可以找到某个教师是某个学生的导师，但得不到该教师的其他信息。相反，对财务部的子模式而言，它需要提供每个教师的工作经历，但不包括学生与导师间的链接关系。这样，财务部可以修改教师的工资，却不能获得该教师指导的学生名单。

9.1.3 数据库管理系统

一个典型的数据库应用涉及多个软件层，我们将其分组成两个主要的层，即应用层和数据库管理层（如图9-2所示）。应用软件处理数据库与用户间的通信，它可能很复杂，用户通过网站访问数据库的应用就是其中一个例子。在这种情况下，整个应用层包括遍及因特网的客户端和使用数据库满足客户端请求的服务器端。

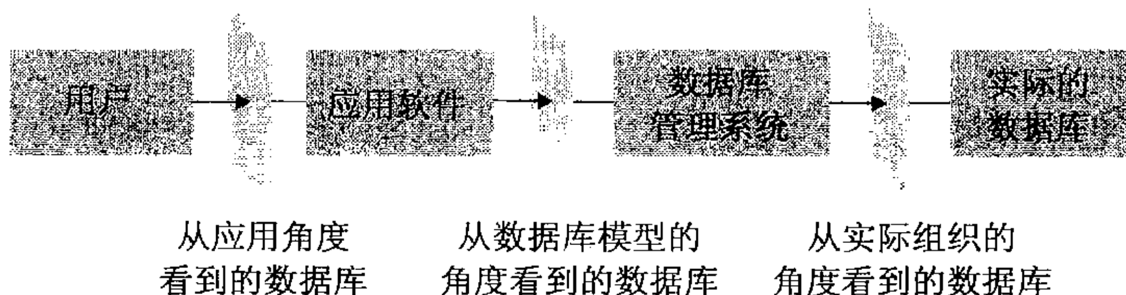


图9-2　一个数据库实现的概念层

分布式数据库

网络能力的提高，促进了分布式数据库的发展，其包含的数据都分别驻留在不同的机器里。例如，一个跨国公司可以将其地方公司的员工记录在本地站点进行存储和维护，然后通过网络链接这些记录，从而创建单个分布式数据库。

分布式数据库包含的数据可以是碎片数据，也可以是数据的副本。前面提到的员工记录的例子就属第一种情况，即数据库的不同片段存放在不同的地方；在第二种情况中，不同的地方存放有数据库同一部分的几个副本，这种副本的存在可以减少信息的获取时间。两种情况都提出了传统的集中式数据库所没有遇到过的新问题：如何掩饰这种数据库的分布式特性，使它像一个连贯的系统那样工作？如何保证数据库更新时，数据库中的各个副本仍保持一致？所以，分布式数据库是当前的一个研究领域。

注意，应用软件并不直接操纵数据库，对数据库的实际操纵由DBMS（database management system，数据库管理系统）来完成。应用软件确定了用户请求的操作以后，就利用DBMS作为抽象工具来得到所需的结果。如果用户要求增加或删除数据，就由DBMS实际更改数据库。如果用户请求检索信息，就由DBMS实际完成所要求的信息搜索。

应用软件与DBMS分离有几个好处。一个好处就是允许构建和使用抽象工具，在软件设计

中我们已反复看到这个重要的简化工作的概念。如果数据库实际是如何存放数据的这样一个细节被DBMS所屏蔽，那么应用软件的设计就可以大大简化了。例如，一个精心设计的DBMS，应用软件无需考虑数据库到底是存放在单台机器里，还是像**分布式数据库**（distributed database）那样，分散存放在一个网络中的许多机器里。DBMS允许应用软件直接访问数据库而不关心数据具体存放的地方，从而很好地处理这些问题。

应用软件与DBMS分离的第二个好处就是，这样的结构提供了对数据库访问进行控制的一种手段。因为这里规定是由DBMS执行对数据库的所有访问，所以DBMS就能实施由不同子模式确定的限制。具体来说，对内部的请求，DBMS能采用整个数据库模式，它也可以要求将每个用户使用的应用软件限制在由该用户子模式描述的范围内。

把用户界面与实际数据库操作分离成两个不同的软件层还有另一个原因，就是可以获得**数据独立性**（data independence），即改变数据库组织本身而不改变应用软件。例如，人事部需要在每个员工记录中增加一个字段，以说明该员工是否参加了本公司新的健康保险计划。如果是由应用软件来直接处理数据库，一个数据格式的变更就要求修改与该数据库有关的所有应用程序。这样一来，原本只需修改人事部的记录，现在却要修改工资计算程序和公司业务通信服务的邮政标签打印程序。

应用软件与DBMS的分离就避免了这种重新编程。为了实现由某个用户提出的对数据库做一项修改的要求，需要改变的只是总体模式，以及涉及这个变更的那些用户的子模式；其他所有用户的子模式都保持不变。因此，基于没有改变的子模式的应用软件，也不必修改。

9.1.4　数据库模型

我们已多次看到如何用抽象来隐藏内部的复杂性，数据库管理系统给出了又一个例子。它隐藏了数据库内部结构的复杂性，使得用户只看到在数据库中是以有效的格式来组织和存储信息的。具体来说，DBMS包含许多例程，它们把按数据库的概念视图表达的命令，转换为实际数据存储系统所要求的操作。这种数据库的概念视图就称为**数据库模型**（database model）。

接下来的几节将讨论关系数据库模型和面向对象的数据库模型。在关系数据库模型中，数据库的概念视图是一组由行和列组成的表格。例如，关于公司员工的信息可以看成这样的一个表格，即每行表示一名员工，各列分别表示姓名、地址、员工代号等。这样，DBMS会包含一些例程，即使在实际信息并没有按行列存放的情况下，也让应用软件从表格的某一行中选取某些项，或者输出工资列中的金额范围。

这些例程构成了应用软件用来访问数据库的抽象工具。更准确地说，通常应用软件用一种通用程序设计语言（这些在第6章讨论过）来编写。这些语言为算法的表达提供了基本元素，但缺少操纵数据库的指令。然而，用这些语言编写的程序可以把DBMS提供的例程作为预先编好的子例程来使用，这实际上扩充了该语言的能力，从而支持了数据库的概念模型。

寻找更好的数据库模型的工作永无止境，其目标就是希望找到的模型能够容易地把复杂的数据库系统概念化，从而能够以简明的方式表达对信息的请求，以及能够产生有效的数据库管理系统。

问题与练习

1. 在一个制造厂中确定两个部门，说明它们对同一个或者类似的库存信息会有不同的用途。然后，说明两个部门的子模式如何不同。
2. 数据库模型的目标是什么？
3. 概述应用软件和DBMS的作用。

9.2 关系模型

本节将更详细地讨论关系数据库模型，它描绘的是用矩形表格存放的数据，这种表格称为**关系**（relation），这类似于电子制表程序显示信息的格式。例如，在关系模型中，一个公司员工的信息就表示为如图9-3所示的关系。

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555
•	•	•	•
•	•	•	•
•	•	•	•

图9-3 包含员工信息的一个关系

关系中的一行称为一个**元组**（tuple）（有人读作“TOO-pul”，也有人读作“TU-pul”）。在如图9-3所示的关系中，元组由某个特定员工的信息组成。因为每列描述的是对应的元组所表示的实体的一些特征或属性，所以关系中的列称为**属性**（attribute）。

9.2.1 关系设计中的问题

设计关系数据库的关键步骤是设计构成这个数据库的关系。尽管这个工作看上去很简单，但对于粗心的设计者来说，仍有不少难以捉摸的陷阱。

假定除了如图9-3所示关系中所包含的那些信息之外，我们还想要添加员工工作的信息。这里需要为每个员工添加一个工作经历，包括如下一些属性：如职务（秘书、办公室经理、楼层主管）、职务代码（每种职务的职务代码是唯一的）、与该职务有关的技能代码、该职务所在部门，以及该员工任职的开始日期和终止日期（如果员工仍任现职，则终止日期用*号表示）等。

解决这个问题的一种方法就是扩展如图9-3所示的关系，在表格中加进这些属性列，如图9-4所示。然而，仔细检查这个结果会发现一些问题。问题之一是，信息的冗余导致了效率低下。这个关系中不再是每个员工对应一个元组，而是每次职务指派就对应一个元组。如果一个员工在公司里历任好几个职务，那么新关系中的几个元组就会包含该员工的相同信息（姓名、地址、员工代号及社会保险号）。例如，因为Baker和Smith担任过多个职务，所以有关他们的个人信息就会有重复。还有，当某个特定的职务由几个员工担任过，那么与此职务相关的部门及相应的技能代码也会在表示职务的每个元组中重复。例如，因为楼层经理由多个员工担任过，所以这个职务的描述就会重复。

Empl Id	Name	Address	SSN	Job Id	Job Title	Skill Code	Dept	Start Date	Term Date
25X15	Joe E. Baker	33 Nowhere St.	111223333	F5	Floor manager	FM3	Sales	9-1-2009	9-30-2010
25X15	Joe E. Baker	33 Nowhere St.	111223333	D7	Dept. head	K2	Sales	10-1-2010	*
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999	F5	Floor manager	FM3	Sales	10-1-2009	*
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S25X	Secretary	T5	Personnel	3-1-1999	4-30-2010
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S26Z	Secretary	T6	Accounting	5-1-2010	*
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•

图9-4 包含冗余的关系

对于这样一种扩展的关系，如果考虑从数据库中删除信息的话，会导致另外一个更为严重的问题。例如，假定只有Joe E. Baker是唯一一个拥有D7这个职务代码的员工，如果他离开公司了，并从图9-4表示的数据库中删除，那么，有关D7的职务信息就会丢失，因为包含D7职务需要K2技能等级这个事实的元组只有与Joe E. Baker有关的那个元组。

PC机的数据库系统

PC机已经在各种应用（从简单到复杂）中广泛使用。在一些基本的数据库应用中，像存放圣诞贺卡清单，或者保存一场保龄球比赛的记录等，因为仅仅是要求能对数据进行存储、打印和排序这样的操作，所以常常只需要用电子表格系统来代替数据库软件就行了。然而，PC机市场上还是有许多数据库系统，例如微软公司的Access数据库。这是9.2节描述过的一个完整的关系数据库，也是图形和报告生成软件。如何运用文中提到的原则来构建今天PC机市场上流行产品的支柱，Access向我们做了很好的诠释。

你也许会认为，能做到只删除元组中一部分信息，就可以解决这个问题，但是这又会引起新的麻烦。比如，F5职务的信息是留存在一个部分的元组中，还是留存在关系中其他什么地方？而且，这种利用部分元组的想法正好说明了该数据库的设计还能够进一步改进。

所有这些问题产生的原因就在于我们在一个单一的关系里融进了多个概念。图9-4中的扩展关系包含了员工的直接信息（姓名、员工代号、地址、社会保险号），有关公司现有职务的信息（职务代号、职务、部门、技能代号），以及有关员工和职务间关系的信息（开始日期、终止日期）。基于以上的分析，我们可以用这样的一种方式来解决问题，即用3个关系来重新设计这一系统，每个关系对应前面的一类信息。我们可以保留图9-3中所示的那个原始关系（现在我们称它为EMPLOYEE关系），再插入称为JOB和ASSIGNMENT的两个新关系，就产生了如图9-5所示的数据库。

EMPLOYEE 关系

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555

JOB 关系

Job Id	JobTitle	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

ASSIGNMENT 关系

Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
23Y34	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

图9-5　由三个关系组成的员工数据库

这样数据库就由3个关系组成，即EMPLOYEE关系包含员工的信息，JOB关系包含职务的信息，ASSIGNMENT关系包含职务经历的信息。其他的信息则隐含在不同关系信息的组合中。例如，如果知道一个员工的代号（也就是员工的ID号），就可以先用ASSIGNMENT关系找到该员工任职过的所有职务，再用JOB关系找到与这些职务有关的部门（如图9-6所示），这样就可以找到这个员工任职过的部门。通过这样一些步骤，任何原先可以从单一的大型关系里面获得的信息，现在都能从3个较小的关系中获得，并且不会出现前面提到的那些问题。

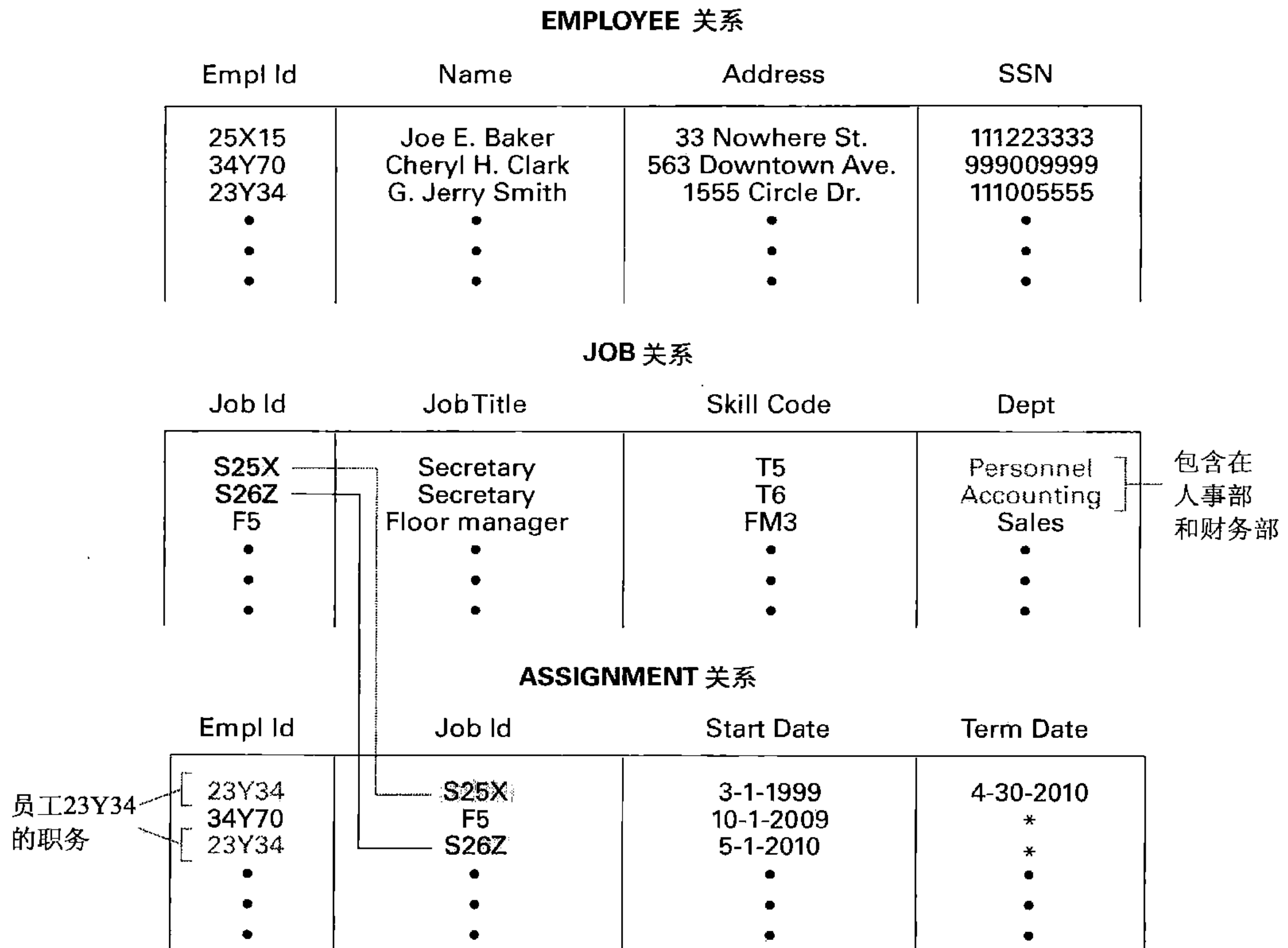


图9-6 查找员工23Y34工作过的部门

但是，把信息划分到不同的关系中，并不总是像上面提到的例子那样顺利。例如，比较图9-7中的原始关系和建议分解成两个关系的EmplId（员工代号）、JobTitle（职务）及Dept（部门）3个属性。乍看起来，双关系系统与单关系系统好像包含相同的信息，但事实并非如此。比如，要查找某员工工作过的部门，这在单关系系统中很容易，只需查找包含该员工代号的那个元组，取出相应的部门即可。然而，在双关系系统中，所要的信息未必存在。我们可以找到该员工的职务及具有这个职务的一个部门，但这并不一定意味着该员工就在这个部门工作，因为几个部门可以有同样的职务。

于是，我们可以看出，把一个关系分解成几个比较小的关系时，信息有时会丢失，有时不会丢失，后者称为**无损分解**（lossless decomposition，或nonloss decomposition）。对这种关系特性的研究是重要的设计依据，其目标就是找出会在数据库设计中引起问题的一些关系特性，并找到重新组织那些关系的方法来消除这些出问题的特性。

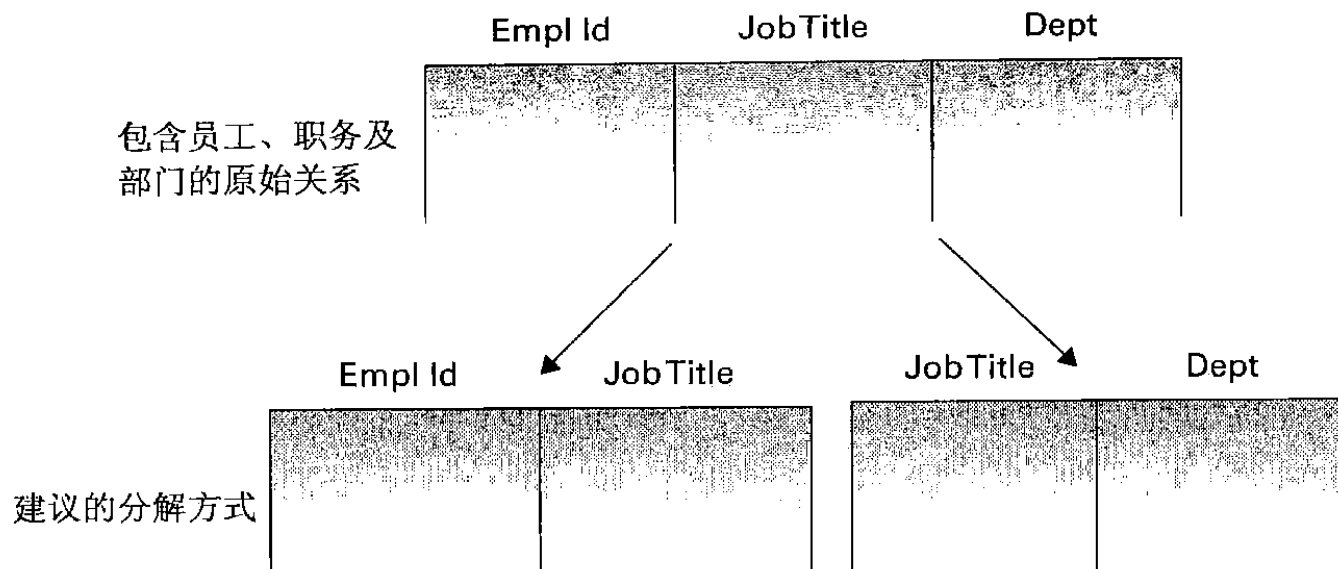


图9-7　关系和提议的分解

9.2.2　关系运算

我们对数据是如何按照关系模型来组织的有了基本的了解以后，接下来的工作就看看如何从由关系组成的数据库中提取信息。我们可以先考察要对关系执行的某些操作。

我们常常会从一个关系中选取某些元组。比如，要检索某个员工的信息，就必须从EMPLOYEE关系中选取包含相应“员工代号”属性值的元组，或者为了得到某一部门的所有职务，就必须从JOB关系中选取具有该部门属性的元组。这样选取的结果是，从父关系中选取的元组构成了另一个关系。选择某一特定员工信息的结果是产生了一个关系，该关系只包含从EMPLOYEE关系获得的一个元组。而选择与某个部门相关的元组，会生成一个关系，其中包含来自JOB关系的几个元组。

简而言之，在一个关系上想要执行的一种运算就是要选取具有某些特性的元组，并把这些选出的元组放到一个新的关系中。为了表示这种运算，我们采用下面的语法：

```
NEW ← SELECT from EMPLOYEE where EmplId = "34Y70"
```

此语句的语义是：创建一个名为NEW的新关系，它包含从EMPLOYEE关系选得的其EmplId属性等于34Y70的那些元组（本例中应该只有一个元组）（如图9-8所示）。

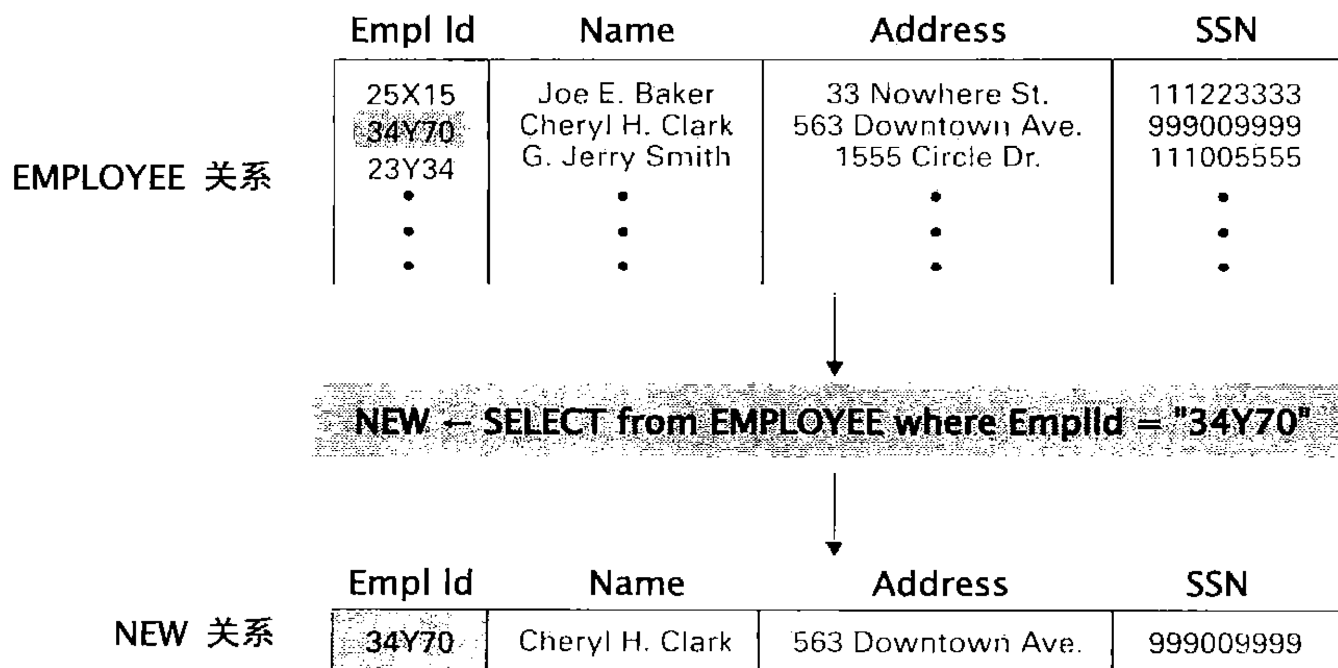


图9-8　SELECT运算

SELECT运算是从一个关系中提取行，与此相反，PROJECT运算则是提取列。例如，假定在查找某部门的职务时，已经从JOB关系中SELECT得到与该部门对应的元组，并把这些关系放

到一个叫NEW1的新关系中。我们查找的清单是这个新关系里的JobTitle列。PROJECT运算就是提取这个列（或者必要时是几个列），并把结果放到一个新关系中。这个运算表示为

```
NEW2 ← PROJECT JobTitle from NEW1
```

其结果是创建另一个新关系（名为NEW2），它包含从NEW1关系中JobTitle列得到的那些值所构成的一个列。

作为PROJECT运算的另一个例子，语句

```
MAIL ← PROJECT Name, Address from EMPLOYEE
```

可以用来获取所有员工的姓名和地址的清单。这个清单是新创建的（有两列的）关系，名为MAIL（如图9-9所示）。

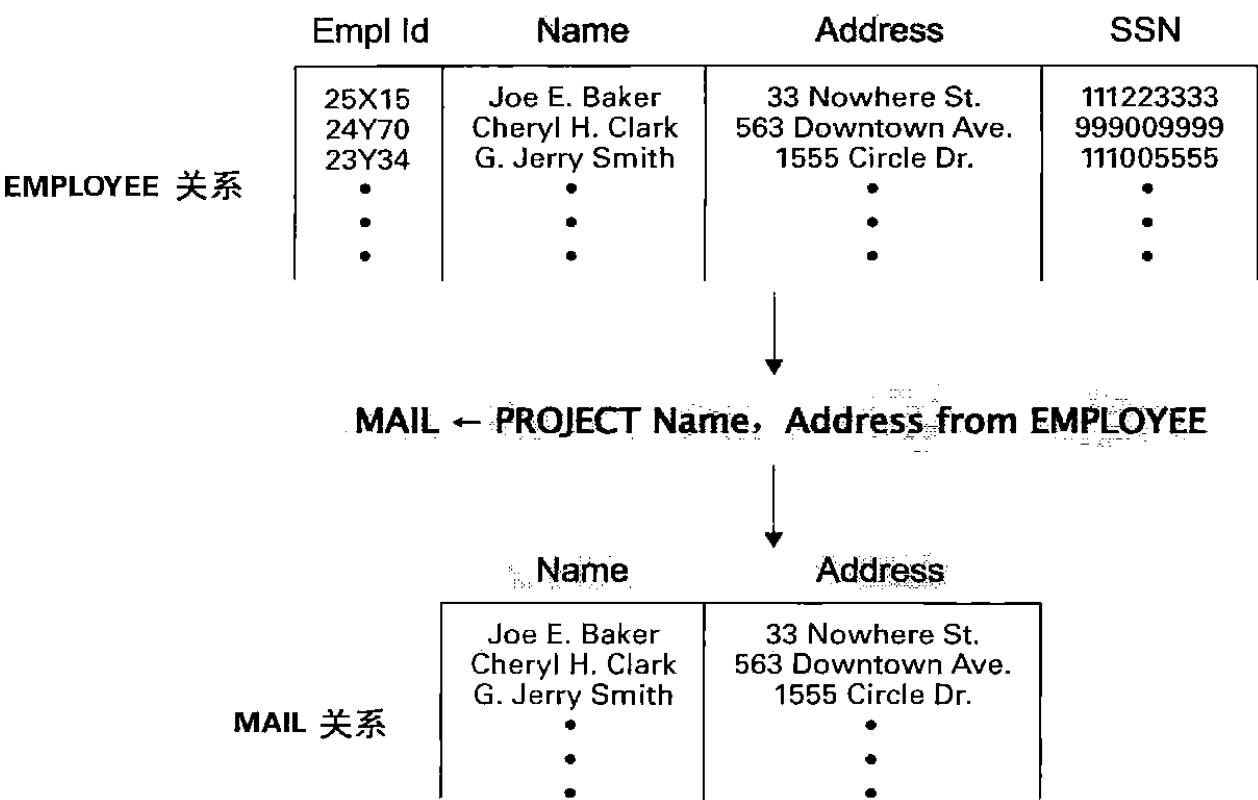


图9-9 PROJECT运算

另外的一个用于连接关系数据库的运算是JOIN运算，它用来把原来不同的关系组合成一个关系。两个关系结合产生一个新关系，而新关系的属性则由原来两个关系的属性组成（如图9-10所示）。这些属性的名称与原先关系中的名称一样，只是每个都加上了原关系作为前缀（如果包含属性V和W的关系A与包含属性X、Y及Z的关系B相结合，那么结果就有名为A.V、A.W、B.X、B.Y和B.Z的5个属性）。这种命名约定保证了新关系的属性只有唯一的名称，即使原先的几个关系中有相同的属性名称也没关系。

新关系的元组（行）由原来两个原始关系的元组串接而成（再见图9-10）。哪几个元组会连接成新关系的元组取决于连接（JOIN）的条件。一个条件就是指定的属性要有相同值。事实上，图9-10表示的就是这种情况，它演示了执行语句

```
C ← JOIN A and B where A.W = B.X
```

的结果。在这个例子里，关系A的一个元组与关系B的一个元组串接，其条件正是两个元组的属性W和X值相等。因此，关系A的元组（r, 2）与关系B的元组（2, m, q）的串接出现在结果中，因为第一个元组中的W属性的值等于第二个元组中X属性的值。另一方面，在最后的关系中并没有关系A的元组（r, 2）与关系B的元组（5, g, p）串接的结果，这是因为这些元组在属性W和X中没有相同的值。

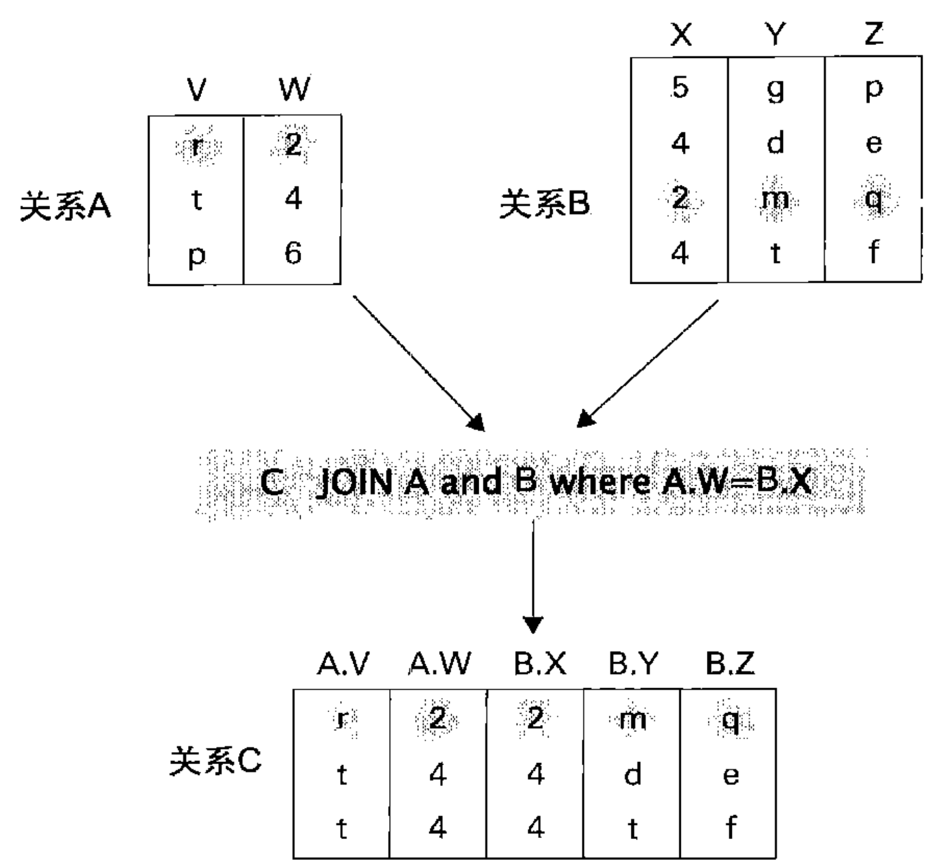


图9-10　JOIN运算

看另一个例子，图9-11所示为执行语句

```
C ← JOIN A and B where A.W < B.X
```

的结果。注意，结果中的元组正是其中关系A中的属性W小于关系B中的属性X的那些元组。

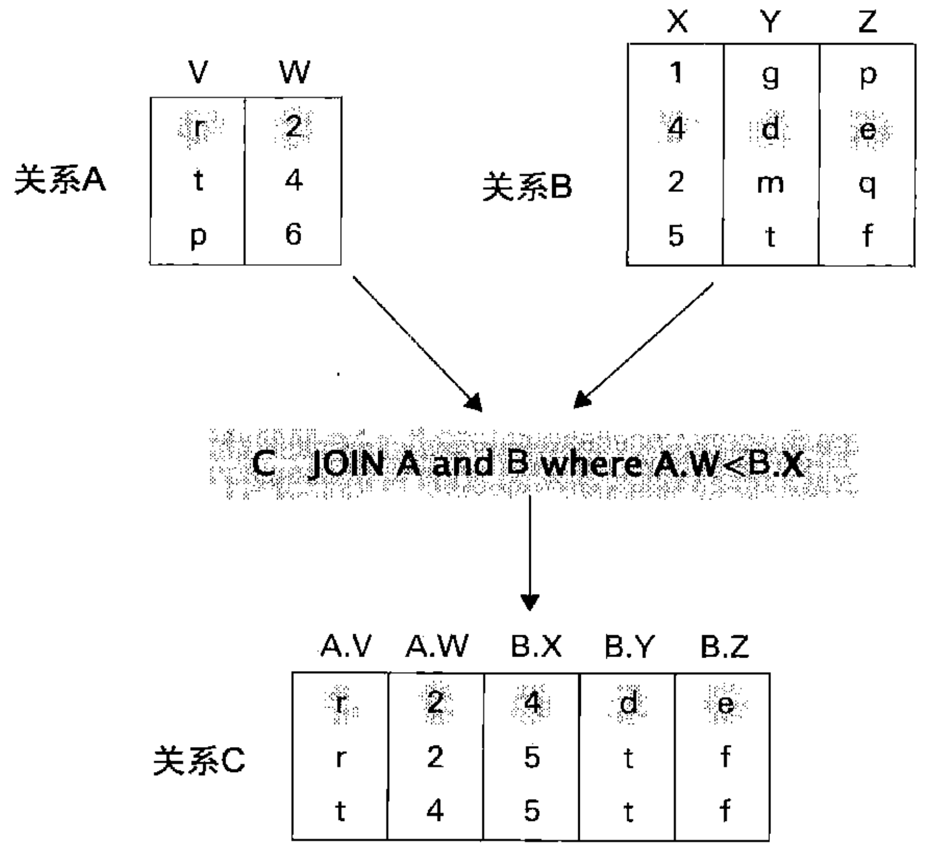


图9-11　JOIN运算的另外一个例子

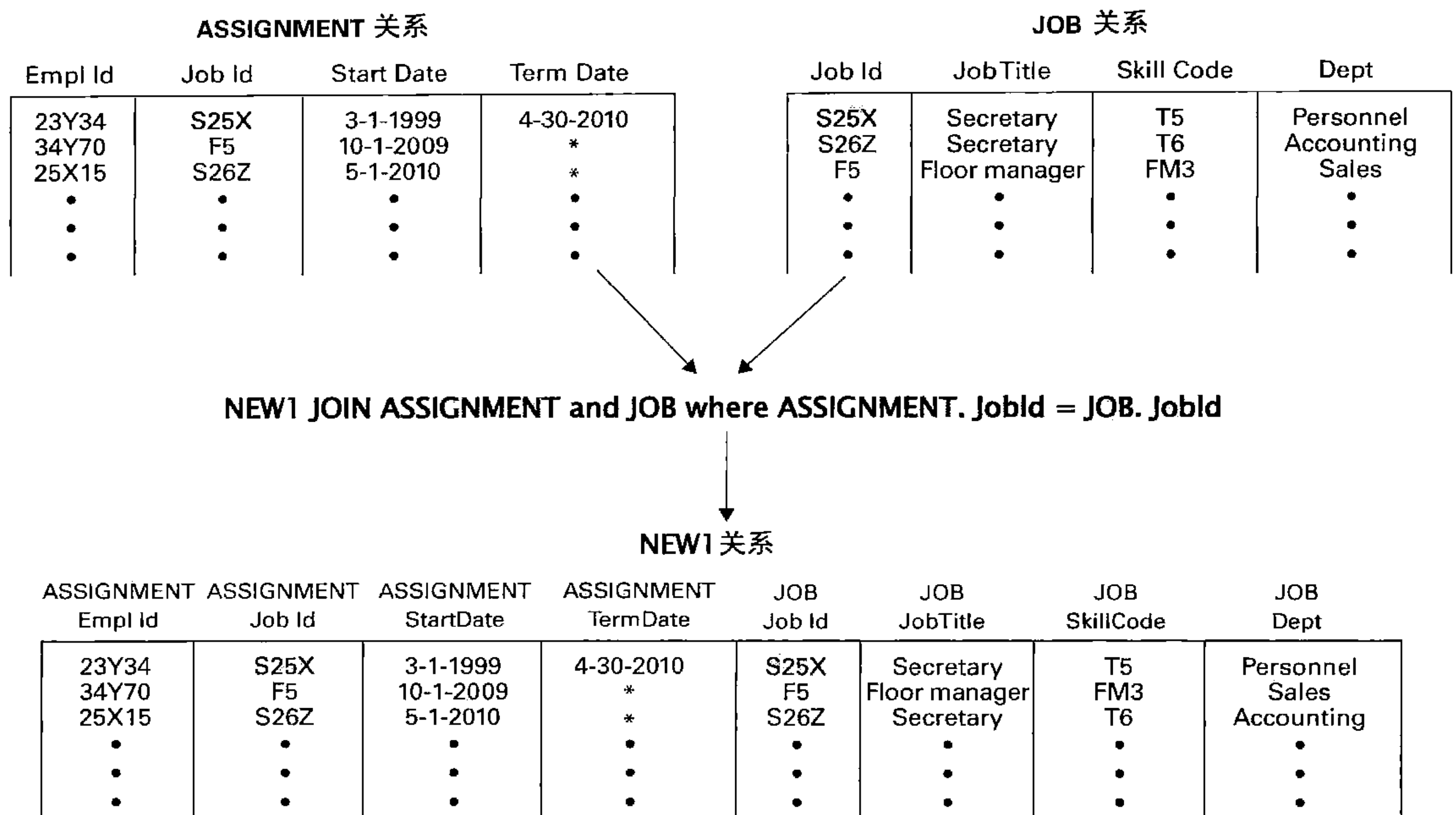
现在来看怎样对图9-5中的数据库用JOIN运算来获取一个清单，这个清单包括所有员工的员工代号及每个员工的工作部门。首先看到，所要的信息分散在一个以上的关系中，所以检索信息单靠SELECT和PROJECT是不行的。实际上，我们所需要的工具是语句

```
NEW1 ← JOIN ASSIGNMENT and JOB where ASSIGNMENT.JobId = JOB.JobId
```

如图9-12所示，它产生了一个关系NEW1。通过这个关系，我们的问题就能得到解决，即先SELECT其中ASSIGNMENT.TermDate等于“*”（“*”表示“员工还在任职期”）的那些元组，然后在ASSIGNMENT.EmplId和JOB.Dept属性上进行PROJECT运算。简而言之，我们需要的信息可以

从图9-5中所示的数据库通过执行以下语句来获得：

```
NEW1 ← JOIN ASSIGNMENT and JOB where ASSIGNMENT.JobId = JOB.JobId
NEW2 ← SELECT from NEW1 where ASSIGNMENT.TermDate = "*"
LIST ← PROJECT ASSIGNMENT.EmplId,JOB.Dept from NEW2
```



ASSIGNMENT 关系

Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
25X15	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

JOB 关系

Job Id	JobTitle	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

NEW1 关系

ASSIGNMENT Empl Id	ASSIGNMENT Job Id	ASSIGNMENT StartDate	ASSIGNMENT TermDate	JOB Job Id	JOB JobTitle	JOB SkillCode	JOB Dept
23Y34	S25X	3-1-1999	4-30-2010	S25X	Secretary	T5	Personnel
34Y70	F5	10-1-2009	*	F5	Floor manager	FM3	Sales
25X15	S26Z	5-1-2010	*	S26Z	Secretary	T6	Accounting
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•

图9-12　JOIN运算的应用

9.2.3 SQL

介绍了基本的关系运算之后，接下来要考虑的是数据库系统的总体结构。我们知道，数据库实际上是存放在海量存储系统中的。为了让应用程序员免于关注这种系统的细节，所以要提供数据库管理系统，使应用软件能够按照数据库模型（如关系模型）来编写。DBMS接受模型方式的命令，并把这些命令转换为与实际存储结构有关的操作。这种转换由DBMS提供的一组例程来实现，应用软件将它们用作抽象工具。这样，基于关系模型的DBMS会包括能够完成SELECT、PROJECT和JOIN运算的例程，应用软件可以调用它们。通过这样的方式，编写应用软件就好像数据库真的是存放在关系模型的一个简单表格中。

今天的关系数据库管理系统不一定提供执行原始形式的SELECT、PROJECT及JOIN运算的例程，而是提供一些组合了这些基本步骤的例程。其中的一个例子就是SQL（Structured Query Language，结构化查询语言），它构成了绝大多数关系数据库查询系统的主干。例如，SQL是因特网上许多数据库服务器采用的关系数据库系统MySQL（读作"My-S-Q-L"）的基础语言。

SQL流行的一个原因是美国国家标准化组织已经将它标准化了，另一个原因是它起初是由IBM公司开发和发布的，因而备受媒体关注，所以流行起来了。本节将解释如何用SQL表达关系数据库的查询。

虽然看起来用SQL表述的查询好像是以命令的形式来完成的，但本质上它是一种陈述性的语句。应当把一条SQL语句看作是对所需信息的一种描述，而不是一串要执行的操作。这样的意义在于，有了SQL，应用程序员不必为开发处理关系的算法而花费精力，他们只要描述所需的信息就可以了。

作为SQL语句的第一个例子，我们现在来考虑上面提到的那个查询例子。在那个例子中，

为了获取所有员工的代号及其所在部门而开发设计了一个3步的处理过程。而在SQL中，整个查询过程用下面这样一条语句就可以表示：

```
Select  EmplId, Dept
from  ASSIGNMENT, JOB
where ASSIGNMENT.JobId = JOB.JobId
  and  ASSIGNMENT.TermDate = '*'
```

从此例可以看到，每条SQL查询语句可包含3条子句，即select子句、from子句和where子句。粗略地说，其实这样一条语句就是请求如下几个操作的结果：from子句中列出的所有关系的JOIN操作，然后是SELECT操作，即选择出满足where子句中条件的那些元组，最后是PROJECT操作，即在select子句中列出的那些元组上进行PROJECT运算。（注意，因为SQL语句中的select子句确定的是PROJECT运算中所用的属性，所以，术语有一些颠倒）。让我们来看一些简单的例子。

语句

```
select  Name, Address
from EMPLOYEE
```

产生了一个包含在EMPLOYEE关系中的所有员工姓名以及地址的清单。注意，这仅仅是一个PROJECT运算。

语句

```
select  EmplId, Name, Address, SSNum
from EMPLOYEE
where Name = 'Cheryl H. Clark'
```

产生了EMPLOYEE关系中与Cheryl H. Clark相关的元组的所有信息。这其实是一个SELECT运算。

语句

```
select  Name, Address
from EMPLOYEE
where Name = 'Cheryl H. Clark'
```

产生了EMPLOYEE关系中Cheryl H. Clark的姓名和地址信息。这是一个SELECT和PROJECT的组合运算。

语句

```
select EMPLOYEE.Name,ASSIGNMENT.StartDate
from EMPLOYEE,ASSIGNMENT
where EMPLOYEE.EmplId = ASSIGNMENT.EmplId
```

产生一个所有员工姓名及其开始工作日期的清单。注意，这是如下几个操作的结果：首先对EMPLOYEE和ASSIGNMENT两个关系进行JOIN操作，然后对在where子句及select子句指定的元组和属性进行SELECT操作和PROJECT操作。

最后，我们需要指出的是，SQL语句除了可以执行查询，还可以定义关系的结构，创建关系，以及修改关系的内容。例如，下面是一些insert into、delete from和update语句的例子。

语句

```
insert into EMPLOYEE
values ('42z12','Sue A. Burt','33 Fair St.','444661111')
```

表示在EMPLOYEE关系中增加给定值的元组，语句

```
delete from EMPLOYEE
where Name = 'G.Jerry Smith'
```

表示从EMPLOYEE关系中把与G. Jerry Smith有关的元组删除掉，而语句

```
update EMPLOYEE
set Address = '1812 Napoleon Ave.'
where Name = 'Joe E.Baker'
```

表示修改EMPLOYEE关系中与Joe E Baker有关的元组中的地址信息。

问题与练习

1. 根据图9-5中所示的EMPLOYEE、JOB和ASSIGNMENT关系提供的部分信息，回答下列问题：
   a. 指出谁既是财务部秘书，又具有在人事部工作的经历？
   b. 指出谁是销售部的楼层经理员？
   c. G. Jerry Smith现在的工作职务是什么？
2. 根据图9-5所示的EMPLOYEE、JOB和ASSIGNMENT关系，写出要获取一份人事部的所有职务清单所需的关系操作。
3. 根据图9-5所示的EMPLOYEE、JOB和ASSIGNMENT关系，写出要获取一份员工姓名及其工作部门清单所需的关系操作。
4. 把第2题和第3题的答案转换成SQL语句。
5. 说明关系模型是如何提供数据独立性的？
6. 说明在一个关系数据库中，不同的关系是怎样联系在一起的？

*9.3 面向对象数据库

另一种数据库模型是基于面向对象范型的。运用面向对象方法构建的数据库称为**面向对象数据库**（object-oriented database），它由对象构成，对象之间通过相互链接来反映它们之间的关系。例如，9.2节中员工数据库的面向对象实现可以包含3个类（对象的类型）：EMPLOYEE、JOB和ASSIGNMENT。EMPLOYEE类的对象可以包含EmplId、Name、Address及SSNum这样一些属性；JOB类的对象可以包含JobId、JobTitle、SkillCode及Dept这些属性；ASSIGNMENT类的对象可以包含StartDate及TermDate这些属性。

面向对象数据库的概念表示如图9-13所示，其中不同对象间的链接可以用相关的对象之间的连线来表示。如果我们注意到EMPLOYEE类的一个对象，可以看出它与ASSIGNMENT类的一组对象关联，这就表示某个员工任职过的不同职务。同样，ASSIGNMENT类的每个对象都与JOB类的一个对象相关联，那么这就表示某个工作与其指派的职务相关。所以，只要沿着表示某员工的对象的链接进行查找，就能找到该员工所有的任职情况。类似地，可以通过查找表示某工作的对象的链接，得到从事过该工作的所有员工的名单。

通常，面向对象数据库中对象间的链接是由DBMS来维护的，所以有关这些链接是如何实现的细节，无需编写应用软件的程序员关心。相反，当向数据库中增加新对象时，应用软件只要指明这个新对象应当与哪些对象相链接就可以了。然后由DBMS创建为记录这些关联信息所需的链接系统。具体地说，DBMS会以类似于链表的形式，把表示某员工的职务的对象链接起来。

面向对象的DBMS的另一个任务就是要为其托管的对象提供永久的储存空间，这个要求看起来显而易见，但它与处理对象的常规方式有着本质的不同。通常，在执行一个面向对象的程序时，程序执行期间创建的对象在程序终止时就会被丢弃。从这个意义上看，可以认为对象是临时的。但是，在数据库中创建或添加的对象，在创建它们的程序终止后必须保存。这样的对象称为**持久**（persistent）对象。所以，创建持久对象与创建常规对象有很大不同。

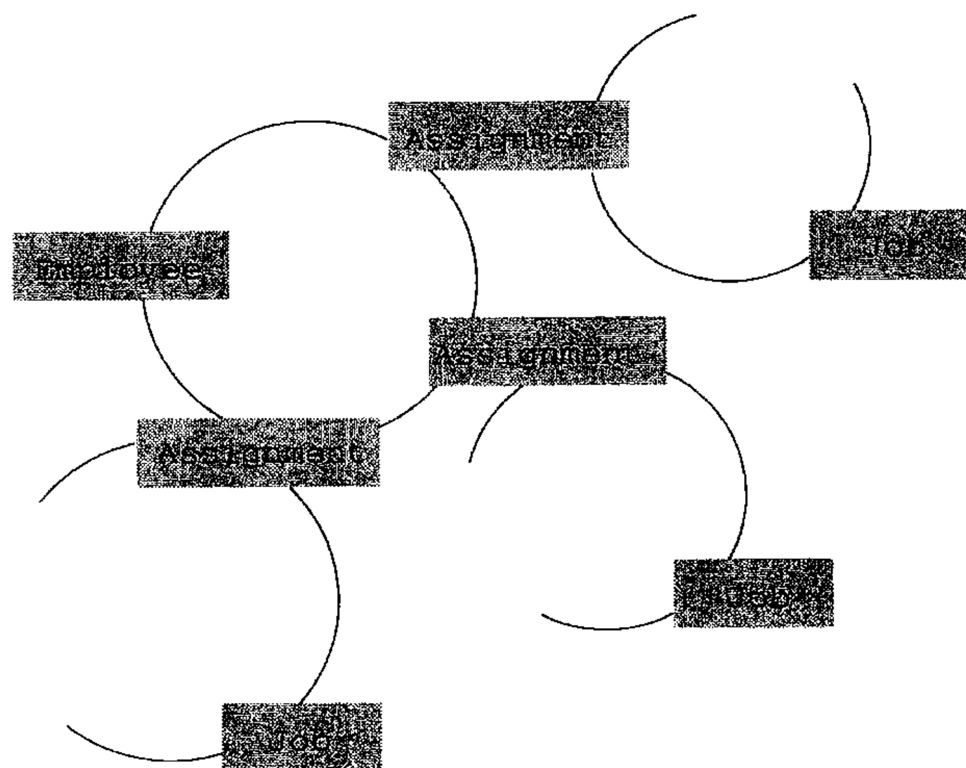


图9-13　面向对象数据库中对象间的关联

面向对象数据库的支持者提出许多论据，来说明为什么用面向对象方法设计的数据库要比用关系方法设计的好。其中一个论据是说，面向对象方法使整个软件系统（应用软件、DBMS和数据库本身）用同样的范型来设计。这与以往开发查询关系数据库的应用软件通常是采用命令型编程语言不同。在这样的任务中，命令范型与关系范型之间的冲突是不可避免的。就我们学习的水平而言，这种差别是很小的，但是多年以来，这种差别正是许多软件错误的根源。即使以现有的知识水平，我们也能够理解，面向对象数据库与面向对象应用程序的结合产生了一种整个系统中的对象相互间进行通信的景象。另一方面，关系数据库与命令型应用程序的结合，给人的感觉是产生了两种本质上不同的组织企图寻找共同接口的景象。

为了理解面向对象数据库相对于关系数据库的另一个优势，这里我们先来看一个在关系数据库中存储员工姓名的问题。如果把全名存放在一个关系的单个属性中，那么仅仅是关于姓氏的查询就比较麻烦了。然而，如果将全名分存于3个分开的属性，即名字、中间名和姓，那么，在处理不遵循这种姓名模式的人员时，又将遇到难题——即使仅处理单一文化中的人名也是如此。在面向对象数据库中，这些问题就可以隐藏在存储员工姓名的对象里。一个员工的姓名可以存储为一个智能对象，它能以不同的格式输出有关员工的姓名。所以，从这些对象外部看，对于处理姓氏、全名、结婚前的姓氏或者绰号等，都是一样地简单。每种视角所涉及的细节都会被封装于对象中。

能把不同的数据格式的术语进行封装也是一个优点。在关系数据库中，关系中的属性是数据库从头至尾需要设计的一部分，所以与这些属性有关的数据类型将遍及整个DBMS。（临时储存的变量必须声明为适当的类型，并且必须要设计出处理不同类型数据的过程。）因此，要扩展一个包含新类型（音频或视频）属性的关系数据库，就很可能会遇到问题。具体来说，贯穿数据库设计的各种过程都必须进行扩展，才能包含这些新的数据类型。然而，在面向对象设计中，用来取得表示员工姓名的对象的过程，同样也可用来取得表示一段影片的对象，这是因为类型的差异被隐藏在所涉及的对象里。因此，面向对象方法显得与多媒体数据库的构建更协调，而这个特性已经被证明具有极大的优势。

面向对象范型对数据库设计而言，还有一个好处，就是它有存储智能对象而不仅仅只是数据的潜力。也就是说，对象能够包含一些方法，用来描述它应如何响应有关它的内容和关系的消息。例如，图9-13中所示的EMPLOYEE类的每个对象都能够包含用于报告和更新这个对象信息的方法，也有显示员工工作经历的方法，甚至还可能有用于更改员工职务的方法。同样，JOB类的每个对象都可以有用于显示职务属性的方法，还可以有用于显示任过此职的员工名单的方

法。这样一来，为了查找员工的工作经历，就没有必要再去编写外部过程来描述如何获得这些信息，而只需要去查询相应的员工对象，就能显示其工作经历了。如果构建的数据库有这样的能力，即它的组成部分能够智能地回应查询请求，那么它就能提供一些传统的关系数据库无法提供的、令人兴奋的功能。

问题与练习

1. 本节讨论的员工数据库中，ASSIGNMENT类的对象的实例应包含哪些方法？
2. 什么是持久对象？
3. 试确定在一个处理仓库货存的面向对象数据库中，要用到哪些类以及这些类的哪些内部特征？
4. 试说明相对关系数据库而言，面向对象数据库的优越性。

*9.4 维护数据库的完整性

个人使用的廉价数据库管理系统是相对比较简单的系统。它们大致有一个单纯的目标，即向用户隐藏数据库实现的技术细节。用这类系统维护的数据库相对比较小，所包含的信息也不会非常重要，信息的丢失或破坏通常只会带来不便，而不至于造成灾难性的后果。当问题真的发生时，用户通常可以直接改正错误项，或者用备份重新恢复数据库，并手工做些必要的修改，使数据库能够及时更新。这样的处理过程也许不太方便，但是，为避免这种麻烦所花的代价要比麻烦本身还大。无论怎么说，这种麻烦只局限于少数人身上，并且经济损失通常也有限。

然而，就大型的、多用户的商用数据库系统来说，利害关系就大得多。数据出错或丢失的代价会十分巨大，甚至会带来毁灭性的后果。在这样的环境下，DBMS的主要作用就是维护数据库的完整性，防止问题的发生，如因某种原因只完成了部分操作，或因疏忽导致不同操作之间相互作用，从而造成数据库信息出错等问题。本节就来阐述DBMS的这种作用。

9.4.1 提交/回滚协议

单个事务，如从一个银行账户向另一账户的转账、预订航班的取消、学生的大学课程的登记，可能在数据库层次需要多个步骤。例如，银行账户间的转账要求一个账户的余额减少，同时另一个账户的余额增加。在这些步骤之间，数据库中的信息可能会不一致。事实上，在第一个账户余额已减少，而另一个账户余额尚未增加的这一短暂时间里，资金有可能会不见。类似地，当为一个乘客重新安排航班座位时，会有一瞬间这个乘客没有座位，或者有一瞬间乘客名单中的乘客数会比实际乘客数多出一位。

对于大型数据库而言，事务量会很繁重。在任意一个瞬间，数据库极有可能处于某个事务的中间状态。一个执行事务的请求或者一个设备的故障，很可能会发生在数据库处于不一致状态的这个时候。

首先我们来考虑故障问题。DBMS的目标就是要保证这种问题不会把数据库冻结在不一致的状态。为了做到这一点，需要维护一个用来记录每个事务活动的日志文件，该日志文件通常存储在磁盘等非易失性的存储系统中。事务被允许更改数据库之前，要执行的更改先被记录到日志文件中。这样，这个日志文件就包含了每个事务操作的持久性记录。

把一个事务的所有步骤记录进日志文件的那个点，称为**提交点**（commit point）。正是在这个点上，DBMS拥有在必要时靠自己重建事务所需要的信息。同时，在这个点上，DBMS在一定意义上为事务提供了保证，即它负责确保事务活动在数据库中得到反映。在出现设备故障的情况下，DBMS可以利用其日志文件中的信息，重建自上一次备份以来已经完成的（提交的）事务。

如果问题出现在事务达到其提交点之前，那么DBMS可能会发现自己不能完成已经执行了一部分的事务。这种情况可以利用日志**回滚**（roll back）（也称为撤销）实际上已被事务执行的活动。例如，在出现故障的情况下，DBMS可以撤销那些在故障发生时没有完成（或称为未提交）的事务，从而恢复原状。

然而，事务的撤销并不局限于设备故障恢复的处理。它们常常也是DBMS正常操作的一部分。例如，由于有试图访问特权信息的情况发生，那么事务可能在完成全部步骤前就会被终止。或者可能是遇到死锁情况，即竞争资源的几个事务发觉自己一直在等待数据，而等待的数据正好被对方使用。在这些情况下，DBMS能够利用日志撤销事务，从而避免了未完成的事务使数据库出错。

为强调DBMS设计的精妙特性，我们要指出，回滚过程中隐藏着许多微妙的问题。一个事务的回滚可能会影响到其他的事务已用过的数据库项。例如，正被回滚的事务可能已经更新了一个账户余额，而另一个事务已进行的活动可能就是基于这个更新的值。这就意味着这些另外的事务也得回滚，从而又会影响到别的事务，结果就产生了称为**级联回滚**（cascading rollback）的问题。

9.4.2　锁定

现在我们来考虑这样一个问题，即执行某个事务时，正值数据库因另一事务而处于变迁状态，这种情况下会无意中造成事务间的相互影响，从而会产生错误的结果。例如，如果一个事务正从一个账户转账到另一个账户，而另一个事务试图计算银行存款总额，就会产生**错误决算问题**（incorrect summary problem）。依据转账步骤的先后次序，结果就可能会造成存款总数不是太大，就是太小。另一个可能出现的问题是**更新丢失问题**（lost update problem）。例如，有两个事务，每个事务都是完成从同一账户扣除金额的操作。如果一个事务读取当前余额时，正值另一个事务刚读取过余额但尚未计算好新余额的时候，于是这两个事务都会在同一个初始账号上进行扣除操作，这样一来，其中一个扣除的结果将不会反映在数据库中。

为了解决这样的问题，DBMS可以强制以一次执行一个整体事务的方式来处理事务，即每个新的事务要进行排队等待，直到它前面的事务全部完成后才能得到执行。但是事务常常要花费很多时间来等待海量存储操作的完成。这里可以采用这样一种方式来解决这个问题，即通过事务之间的交叉执行，可以实现把一个事务等待的时间分配给另一个事务，用来处理它已经获得的数据。大多数大型数据库管理系统都有一个调度程序来协调事务间的分时，这非常像多道程序操作系统里协调进程的交叉处理（参见3.3节）。

为了防止错误决算问题和更新丢失问题这一类异常情况的出现，这些调度程序都包含了一个**锁定协议**（locking protocol），该协议规定，数据库中当前正在被某个事务使用的项目都要加以标记。这些标记称为锁，已标记的项目称为被锁定。有两种类型的锁比较常见，即**共享锁**（shared lock）和**排它锁**（exclusive lock），它们分别对应于访问事务所需数据的两种形式，即共享访问和互斥访问。如果一个事务不会改变数据项，那么它要求的就是共享访问，这就意味着允许其他事务看到该数据项。而如果一个事务要改变数据项，那么它要求的就必须是互斥访问，这就意味着，只有该事务才能访问该数据项。

在锁定协议中，每次事务请求访问数据项时，它还必须告诉DBMS它所需的访问类型。如果事务请求的是对一个数据项的共享访问，那么不论这个数据项是否用共享锁锁定，这个访问都将得到批准，并且将该数据项用共享锁进行标记。但是，如果被请求的数据项已经用排它锁标记了，那么别的访问都将会被拒绝。如果事务对数据项的访问要求是互斥访问，那么只有在这个数据项没有被锁定的情况下，请求才能被批准。通过这样一种方式，一个准备改动数据项的事务就可以通过获得的互斥访问，来防止别的事务对该数据项的干预。反之，如果几个事务都不会改动数据项，那么它们就能够对这个数据项实现共享访问。当然，事务完成了对数据项的访问以后，它就会通知DBMS，并解除相关的锁定。

当出现事务的访问请求被拒绝的情况时，可以用许多不同的算法进行处理。一种算法就是强制事务等待，直至所请求的项可用为止，但是这种方法容易造成死锁。因为两个事务要求对同样的两个数据项进行互斥访问的时候，如果每一个事务都获得了对其中一个数据项的互斥访问权限，并且又坚持等待另一个数据项，那么它们就会出现阻塞情况。为了避免这种死锁的发生，有些数据库管理系统会让较早的事务优先处理。也就是说，如果一个较早的事务要求访问被稍后的事务锁定的数据项时，那么就强制那个稍后的事务释放其所有的数据项，而它的活动都会被撤销（依据日志文件）。于是，较早的事务就获得了对所需数据项的访问权限，而稍后的事务只得重新开始。如果这个稍后的事务一直被抢占，那么随着过程的进展它也会变老，最终成为一个具有较高优先级的老事务。这个协议，称为**受伤等待协议**（wound-wait protocol）（老的事务将新的事务挂起，而新的事务等待变成老的事务），这样就能保证每个事务最终都能完成它的任务。

问题与练习

1. 说明事务到达了它的提交点与没到达提交点的区别是什么。
2. DBMS是怎样防止大量的级联回滚的？
3. 假定一个账户的初始余额是400美元。有两个事务，一个事务从这个账户中支取100美元，另一个事务也从同一账户中支取200美元。请说明，这两个不加控制的交叉事务怎样才能使账户的最终余额为100美元、200美元和300美元。
4. a. 简述事务对数据库中的数据项请求共享访问的可能结果。
   b. 简述事务对数据库中的数据项请求互斥访问的可能结果。
5. 试描述会导致执行数据库操作的事务间出现死锁的一系列事件。
6. 请说明怎样打破第5题中的死锁情况。你的解决办法是否要用到数据库管理系统中的日志文件？请解释你的答案。

*9.5 传统的文件结构

本节我们抛开多维数据库系统的研究来讨论传统的文件结构。这些结构代表了数据存储和检索系统的历史开端，现在的数据库技术就是由此发展而来的。为这些结构开发的许多技术（如索引技术和散列技术等）是构建今天大规模、复杂数据库的重要工具。

9.5.1 顺序文件

顺序文件（sequential file）是这样的一种文件，即它从头到尾都是以顺序的方式进行访问的，好像文件中的信息都排成一行。这种文件的例子有音频文件、视频文件、包含程序的文件和包含文本文档的文件等。事实上，大多数由个人计算机用户创建的文件都是顺序文件。例如，当保存一个电子表格时，它的信息就会作为一个顺序文件进行编码和保存，电子表格应用软件能够重新构建电子表格。

文本文件是顺序文件，它的每个逻辑记录是用ASCII码或Unicode码编码而成的单个符号。文本文件常常作为一种基本的工具，用来构建诸如员工记录文件这些更为复杂的顺序文件。为此，只需建立一个统一格式，把每个员工的信息表示为一串文本，然后按照格式对这些信息进行编码，接下来就把这些员工记录一个接一个地记录成一个文本串。例如，可以构建这样的一个简单的员工文件，即每个员工记录为可以输入31个字符的字符串，其中25个字符作为一段，用来表示员工的姓名（每段中多余处用空格填充），随后6个字符作为一段，用来表示员工的工号。最终的文件将会是一个很长的编码过的字符串，其中每31个字符组成的字符块代表了一个

员工的信息（如图9-14所示）。从文件中，我们可以根据由31个字符的信息块所组成的逻辑记录来实现信息检索，每个块中的各个字段是根据构成块的统一格式来识别的。

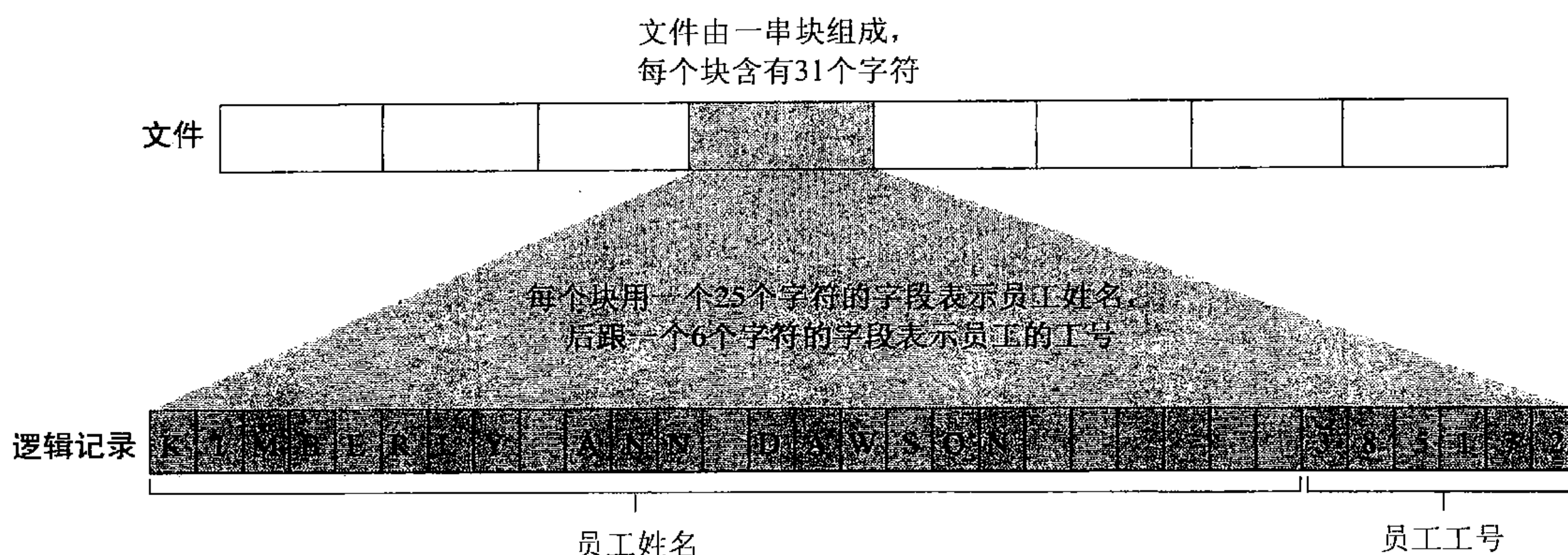


图9-14　以文本文件实现的一个简单的员工文件结构

顺序文件中的数据在大容量存储器里存放时必须要保持文件的顺序特性。如果大容量存储系统本身具有顺序性（如磁带和CD），那么就可以直接做到这一点。在此，我们只需根据存储介质的顺序特性，将文件记录到存储介质中。然后，处理文件的过程仅仅是：按照文件内容建立的顺序来读取和处理它们。播放音乐CD就是这么一个过程，因为音乐作为一个顺序文件，沿着螺旋型轨道，一个扇区接着一个扇区进行存放。

然而，在磁盘存储的情况下，文件将分散在不同的扇区中，因而会以各种顺序来读取。为了保持正确的顺序，大多数操作系统（更准确地说是文件管理程序）都会维护一张存放文件的扇区列表。这个表，作为磁盘目录系统的一部分与文件记录在同一磁盘上。即使文件实际上分散存放在磁盘的不同部分，但是利用这个表，操作系统就能以正确的顺序检索扇区，就好像文件真的是按顺序存放一样。

顺序文件处理中的一个固有问题就是必须要检测何时到达文件的末尾。通常我们把顺序文件的末尾称为**EOF**（End-Of-File，文件结束）。有许多方法可以用来标识EOF，一种方法是在文件的末尾放置一个专用的标记，称为**哨兵**（sentinel）。另一种方法是利用操作系统的目录系统中的信息来确定一个文件的EOF。也就是说，由于操作系统知道哪个扇区包含有此文件，它也就知道这个文件在什么地方结束。

一个小公司的工资单处理就是使用顺序文件的一个典型例子。这里我们可以想象出一个顺序文件，它是由一系列的逻辑记录组成，每条记录都包含一个员工的薪水信息（如姓名、员工工号、工资等级等）。依据这些信息就能定期打印出支票，每读取一个员工的记录，就能计算出该员工的工资，然后再打出相对应的支票。处理这样一个顺序文件的操作，可由以下语句做示例：

```
while (未到达EOF ) do
(从该文件中提取下一条记录并处理它)
```

当顺序文件中的逻辑记录用键字段来标识时，文件通常就可以这样安排，即按照由键（可能是字母键或者是数字键）确定的顺序来安排文件中的记录。这样一种安排简化了文件信息的处理工作。例如，假定处理工资时，必须要求依据考勤单的信息更新每个员工的记录。如果包含考勤单记录的文件和包含员工记录的文件都根据同一个键按照同样的次序存放，那么，就能顺序地访问两个文件来进行更新处理，即用从一个文件读取的考勤单来更新另一个文件的相应记录。这是一个重大改进，因为如果文件不按照相应次序来存放，就必须反复地查找，而上述方法就克服了

这个缺点。所以，更新典型的顺序文件通常需要多个步骤进行处理。首先，新信息（例如考勤单中的信息）记录在一个称为事务文件的顺序文件中，这个事务文件按照要被更新的文件（称主文件）的次序进行排序，然后，通过从两个文件中顺序地读取记录来对主文件记录进行更新。

与这种更新过程稍有不同的是归并过程，即把两个顺序文件合并成一个包含原来两个文件记录的新文件。假定两个输入文件的记录是依据一个公共的键字段按照升序来排列的，并假定归并产生的输出文件也是按键的升序来排列。图9-15概述了这个典型的归并算法。其基本思想是顺序地扫描两个输入文件，从而构建出输出文件（如图9-16所示）。

```
procedure MergeFiles (InputFileA , InputFileB ,OutputFile)
if (两个输入文件都处于EOF) then (停止，OutputFile为空)
if (InputFileA，不在EOF) then (声明它的第一个记录为当前记录)
if (InputFileB，不在EOF) then (声明它的第一个记录为当前记录)
while (两个输入文件都不在EOF) do
   (将键字段值较小的当前记录放在OutputFile中；
  if (该当前记录是其对应输入文件的最后一个记录)
    then (声明该输入文件在EOF)
    else (声明该输入文件中的下一个记录是该文件的当前记录)
  )
从不在EOF的输入文件的当前记录开始
  复制其余记录到OutputFile
```

图9-15 归并两个顺序文件的过程

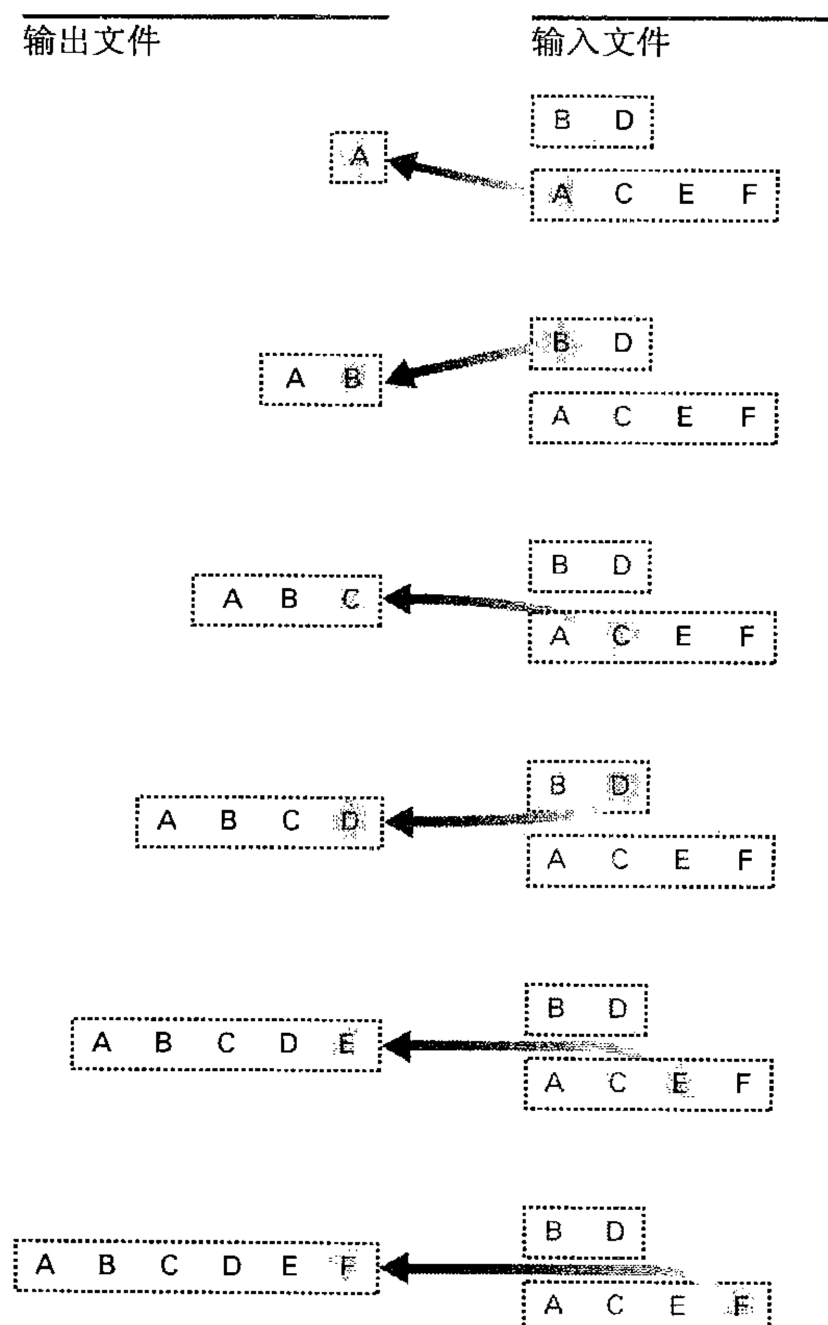


图9-16 归并算法的应用（字母用于代表整条记录，具体字母表示记录的键字段的值）

9.5.2 索引文件

对于数据处理的次序就是其文件存储次序的情况，顺序文件是存储这类数据的理想选择。然而，当文件必须以一种不可预测的次序进行检索时，那么这种文件的效率就不高了。在这种情况下，需要一种方法来快速确定所需逻辑记录的位置。一种流行的方法就是使用文件索引，这种方式与书本里的索引用来定位主题在书中位置的方式非常一致。这种文件系统称为**索引文件**（indexed file）。

文件的索引包含存放在该文件中的键的列表和指示包含每个键的记录存放位置的项。这样一来，为了要找到某个记录，首先需要在索引中找到指定的键，然后再读取存放在与该键相关位置处的信息块。

文件的索引通常作为一个单独的文件与被索引的文件存放在同一个大容量存储设备里。在文件处理开始之前，通常要先将索引调入主存储器中，这样一来，当需要访问文件中的记录时，就会很容易地找到该记录（如图9-17所示）。

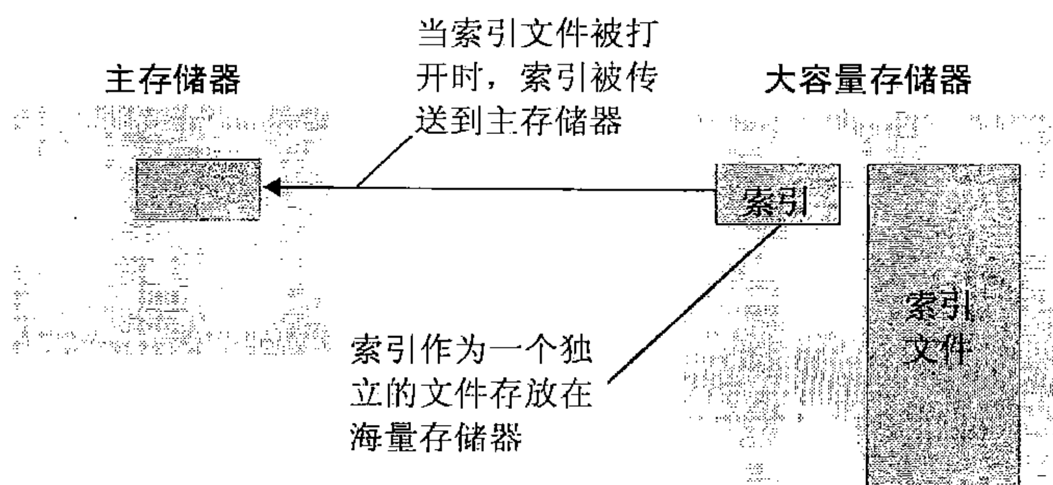


图9-17 打开索引文件

在维护员工记录时就有索引文件的一个典型例子。当想检索一个员工的记录时，如果使用索引就可以避免冗长的查找操作。具体来说，如果员工记录文件用员工工号进行索引，那么只要知道员工的工号，就能很快查到该员工的记录。另一个例子是音乐CD的播放，如果利用索引就能较快地访问到各首乐曲。

多年以来，在基本索引概念的基础上，已经使用了许多不同的索引技术。构建索引的一种方式是运用层次化的方式，以便索引呈现出分层结构或树结构。最突出的例子就是大多数操作系统为组织文件存储所采用的分层目录系统。在这种情况下，目录（即文件夹）起到索引的作用，而每个索引又包含了指向其子索引的链接。从这个角度来看，整个文件系统只是一个大型的索引文件。

9.5.3 散列文件

尽管索引技术为访问数据存储结构中的数据项提供了一种较快的访问机制，但维护索引的开销也比较大。**散列**（hashing）技术也能提供类似的访问效果，但无须那样大的开销。与索引系统的情况一样，散列技术也是利用键值来定位记录。但是散列技术并不是从索引中查找键，而是直接通过键确定记录的位置。

散列系统可以概括如下：数据存储空间被分成几个区，称为**存储桶**（bucket），每个桶能放几条记录。根据将键的值转换为桶号的算法，可以将记录分散存放于这些桶中。这里，将键的值转换为桶号的算法称为**散列函数**（hash function）。每条记录就存放在通过这样处理确定的桶里。因此，要检索一条已经置于这种存储结构中的记录，首先要对该记录的标识键应用散列函

数，以确定相应的桶，然后检索桶中内容，最后从检索的数据中查找所需要的记录。

散列不仅能用于从大容量存储器中检索数据，也是从存放在主存的大数据块中检索数据项的一种方法。当散列用在大容量存储器中的存储结构时，其结果称为**散列文件**（hash file）。当散列用在主存中的存储结构时，其结果通常称为**散列表**（hash table）。

通过散列法认证

散列法不只是用来作为构建高效数据存储系统的一种手段。例如，散列法还可以用作认证因特网上传送的消息的一种方法。其基本思想是：以秘密方式对消息进行散列运算，然后将得到的值与消息一起传送。为了认证消息，接收方对收到的消息进行散列处理（以同样的秘密方式），并确认得到的值与原始值是否一致（这里假定经过散列处理后得到相同的值，而消息发生改变的可能性很小）。如果得到的值与原来的值不一致，就认为该消息已被破坏。那些对此感兴趣的人可能希望从因特网上搜寻到有关MD5的信息，其实MD5是在认证应用领域有着广泛应用的散列函数。

错误检测技术可以看作散列法在认证领域的一种应用，其实已经是一目了然的事了。例如，校验位的使用本质上就是一个散列系统，在此系统中，位模式只散列为0和1，然后将这个值与初始位模式一起传送。如果最终接收的位模式不能散列成同样的值，那么就认为这个位模式已被破坏。

现在我们把散列技术运用在典型的员工文件中，这里，每条记录包含的是公司中一个员工的信息。首先，在海量存储器中创建几个可用的区域，用来实现桶的功能。至于如何设计决定桶的数目和每个桶的大小，稍后再讨论。现在，我们假定已创建了41个桶，桶号从0一直到40。（我们选择41个桶，而不是偶数40个桶，原因稍后再解释。）

现在我们假设用员工工号来作为识别员工记录的键。这样，下一步工作就是设计一个散列函数，把这些键转换成桶号。虽然员工工号可能是25X3Z或J2X35这样的格式，不是数字型的，但它们是以位模式存储的，我们能够将这些位模式解释成数字，利用这个数字解释，就能让任何一个键去除以可用的桶号，然后记下余数。在这个例子中，余数将是0~40的一个整数。因此，我们就可以用每次做除法得到的余数来确定41个桶中的一个（如图9-18所示）。

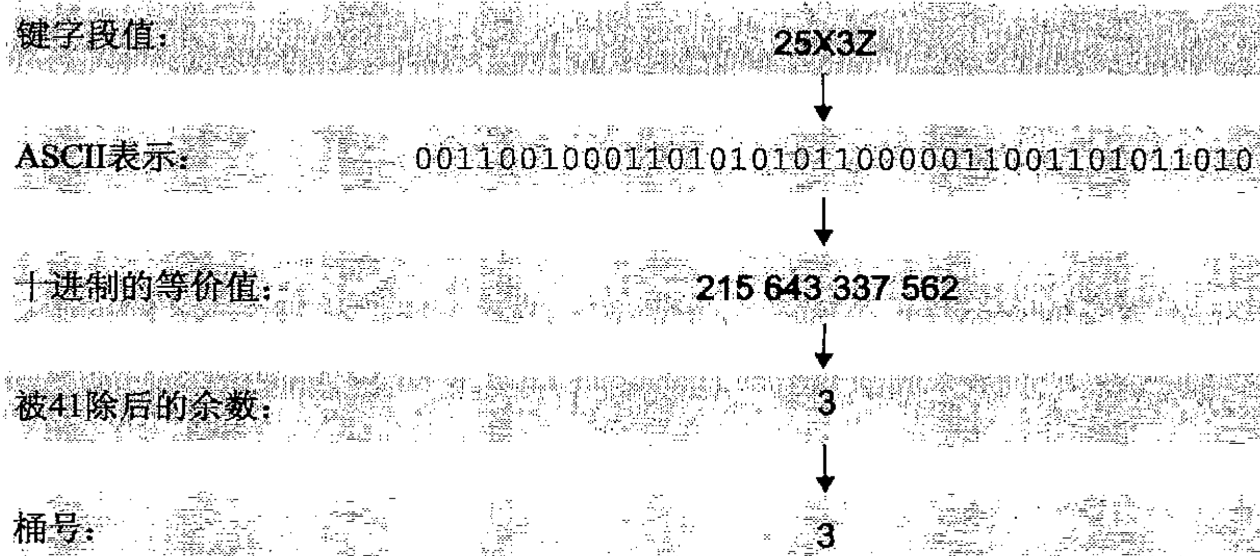


图9-18 将键字段值25X3Z散列到41个桶中的一个

以此作为我们的散列函数，接下来再分别考虑每条记录，继续构建文件。通过使用散列函数对其键除以41得到一个桶号，然后再把该记录存放在这个桶中（如图9-19所示）。以后，如果当我们需要检索一条记录时，只需将这个散列函数应用到该记录的键，以确定相应的桶号，然

后就可以从这个桶里查找所要的记录。

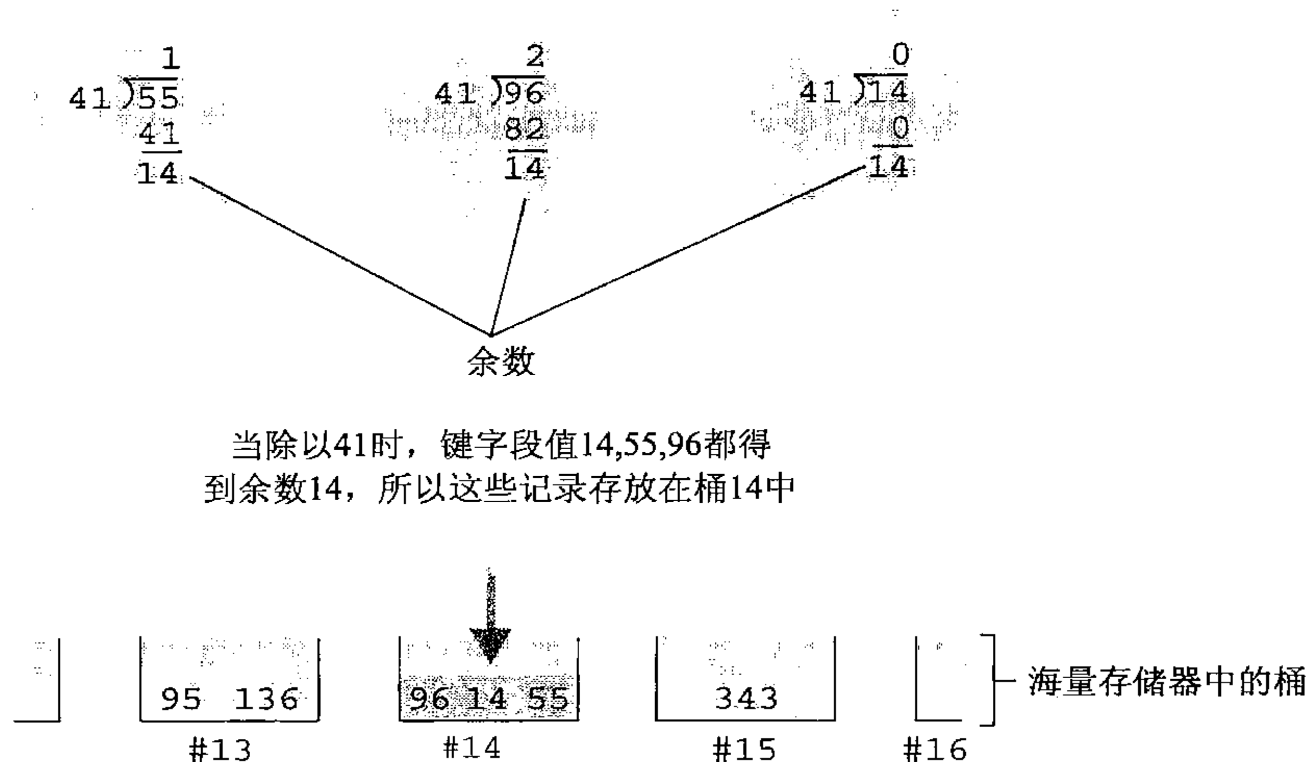


图9-19　散列系统的基本原理

现在，让我们来重新考虑一下把存储区分成41个桶的问题。首先注意，要想得到一个有效的散列系统，要存放的记录应当均匀地分布在这些桶中。如果发生一个不成比例的键数目恰巧散列到同一个桶里——这种现象被称为**群集**（clustering）——那么，在一个桶里就会存入不成比例数目的记录。结果是，从这个桶里检索一条记录就会花费更多的查询时间，这也就失去了散列技术的优势。

现在再来看，如果我们选择把存储区域分成40个桶，而不是41个桶，那么散列函数涉及的除数（即除键的值）就是40，而不是41。但是，如果被除数和除数有一个公因子，而这个公因子也会出现在余数中。具体来说，如果存储在散列文件中的数据项的键碰巧都是5的倍数（也是40的约数），那么当用40来除时，5这个因子就会出现在余数中，并且数据项就会群集到与余数0、5、10、15、20、25、30及35相对应的那些桶里。类似的情况还会出现在键是2、4、8、10及20的倍数的情形中，因为它们也都是40的约数。因此，我们选择把存储区域分成41个桶，因为41是素数，选择它，就可以消除公约数，从而减少群集的可能性。

但是，群集的可能性绝对不能完全消除，即使用的是精心设计的散列函数，在文件构建过程的早期，还是非常有可能存在两个键经过散列后，得到同一个值的情况。这个现象被称为**碰撞**（collision）。为了理解其中的原因，考虑下面的情况。

假设我们已经建立了一个在41个桶中随机分配记录的散列函数，这时候的存储系统是空的，并且准备一次插入一条记录。当插入第一条记录时，它将会被放进一个空桶里。然而，当插入第二条记录时，41个桶中还有40个桶是空的，这样一来，第二条记录被放进空桶的概率只有40／41。假设第二条记录被放进了一个空桶，那么当放第三条记录时就只能找到39个空桶，所以，被放进空桶的概率是39／41。继续这个过程，就可以发现，如果前7条记录都被放进了空桶，那么第八条记录被放进余下空桶的概率就只有34／41。

基于以上的分析，我们就能计算出所有前8条记录都被放进空桶的概率，即每条记录被放入空桶概率的乘积，这里假设前面的记录都已被放入空桶。那么这个概率为

$$(41/41)(40/41)(39/41)(38/41)\ldots(34/41)=0.482$$

问题是这个结果小于一半。也就是说，当在41个桶里分配记录时，很可能在存放第八条记录的时候，就会发生碰撞现象。

发生碰撞的高概率表明，不管如何精心选择散列函数，设计任何一个散列系统时都必须要考虑到群集现象。特别是，一个桶有可能会装满或者溢出。这种问题的一种解决方法就是，允许扩展桶的大小。另一种解决方法是，允许桶溢出到一个专门为解决这种问题而保留的溢出区。无论如何，群集情况和溢出情况的出现都将使散列文件的性能明显降低。

研究表明，作为一般规律，只要记录的数目与文件中总的记录容量之比——将这个比率称为**负载因子**（load factor）——保持在50%以下，那么散列文件就会表现出良好的性能。但是，如果负载因子攀升至超过75%，那么系统的性能通常就会降低（严重的群集现象会造成有些桶装满或者可能溢出）。由于这个原因，如果散列存储系统的负载因子接近75%这个值，那么它通常会以一个更大的容量进行重建。最后得出结论，通过实现散列系统来获得记录检索的高效率是需要花费一定代价的。

问题与练习

1. 依据图9-15所示的归并算法，假设一个输入文件包含键字段值等于B和E的记录，而另一个输入文件包含A、C、D和F，试归并这两个文件。
2. 归并算法是一种称为归并排序的流行排序算法的核心。你能否说明这种算法？（提示：任何非空的文件可以看成是单项文件的集合。）
3. 文件是顺序的，这是物理属性还是概念属性？
4. 从索引文件中检索记录需要哪些步骤？
5. 试说明：如果选取了不恰当的散列函数，其散列存储系统的性能不比顺序文件优越。
6. 假设一个散列存储系统是用文中提到的除法散列函数构建的，但是这里只用6个存储桶。对以下各键值，确定相应记录应该放进哪个桶。会出现什么问题？为什么？
   a. 24　b. 30　c. 3　d. 18　e. 15
   f. 21　g. 9　h. 39　i. 27　j. 0
7. 要想出现两个人的生日在同一天的可能性，至少需要多少人？试说明这个问题与本节内容有何关系？

9.6 数据挖掘

一个迅速发展的并与数据库技术紧密相关的学科就是数据挖掘，它包括了在数据集上发现模式的技术。数据挖掘已经成为许多领域的重要工具，包括市场营销、库存管理、质量控制、借贷风险管理、欺诈检测和投资分析等。数据挖掘技术甚至可以运用于那些似乎不大可能会用到的场合，例如用于确定某些以DNA分子进行编码的基因功能以及描述有机组织的特性。

数据挖掘活动与传统的数据库查询不同，原因在于数据挖掘所做的工作是设法确定以前未知的模式，而传统数据库需要做的只是检索已经存储好了的事实。此外，数据挖掘操作的是静态的数据集合，称为**数据仓库**（data warehouse），而不是经常要更新的“联机”运行的数据库。这些仓库往往是数据库或数据库集的“快照”。因为静态系统中寻找模式要比动态系统中简单，所以用它们来替代实际运行的数据库。

还需要注意的是，数据挖掘的主题不单局限于计算领域，而且还涉及统计学领域。事实上，很多人认为，由于数据挖掘源自于试图对大量不同的数据集进行统计分析，因而它更像是统计学的一种应用，而并非计算机科学的一个领域。

数据挖掘有两种常见的形式：**类型描述**（class description）和**类型识别**（class discrimination）。类型描述用来找出描绘一组数据项的属性，而类型识别用来找出区分两组数据项的属性。例如，类型描述技术可以用来发现购买经济型轿车的人的特点，而类型识别技术可以用来发现能区分

买二手车与买新车的顾客的特性。

生物信息学

数据库技术和数据挖掘技术的进步，扩展了生物学家在涉及模式识别和有机化合物分类研究领域可使用的工具。结果就产生了生物学的一个新领域，称为生物信息学。现在的生物信息学源于对DNA解码的研究工作，它包括了如蛋白质分类和理解蛋白质相互作用序列（称为生物化学路径）的这样一些研究。虽然通常认为生物信息学是生物学的一个部分，但它很好地例证了计算机科学是如何影响甚至扎根于其他领域的。

另一种数据挖掘的形式是**聚类分析**（cluster analysis），它用来以发现类型。注意，这与类型描述不同，类型描述是用来发现已经确定的类型中成员的属性。更明确地说，聚类分析试图找到那些能引导发现组群的数据项的特性。例如，在分析观看某部电影的观众年龄信息的过程中，通过聚类分析可能会发现，观众会分成两个年龄组，即4～10岁一组和25～40岁一组。（也许这影片吸引了孩子和他们的父母？）

还有一种数据挖掘的形式，称为**关联分析**（association analysis），它的工作是寻找数据组之间的联系。要找到既买土豆片又买啤酒和饮料的顾客，或者在正常的工作日购物又能享受退休优惠的顾客，正好可以使用关联分析。

孤立点分析（outlier analysis）是数据挖掘的另一种形式，它试图识别出不符合规则的数据项。孤立点分析可以用于确定数据集中的错误，它还可以检测信用卡，如果发现信用卡突然偏离客户的正常消费模式，那么就可以确定该信用卡被盗用，甚至可以通过发现反常的行为识别出潜在的恐怖分子。

最后，还有一种数据挖掘形式，称为**序列模式分析**（sequential pattern analysis），它试图确定随时间变化的行为模式。例如，序列模式分析可以揭示股票市场等经济系统中的趋势，或者气候环境等环境系统中的趋势。

最后这个例子表明，数据挖掘的结果可以用来预测未来的行为。如果一个数据项具有表征某个类型的属性，那么这个数据项就可能表现为这个类型的成员。然而，许多数据挖掘项目只是旨在获得对数据的更好理解，如利用数据挖掘来解开DNA之谜。无论如何，数据挖掘具有巨大的潜在应用领域，并且有望成为未来一个活跃的研究领域。

注意，数据库技术和数据挖掘之间的关系就像堂兄弟一样，一个领域的研究成果在另一个领域也会有反映。数据库技术广泛运用，使得数据仓库具有以**多维数据集**（data cube，从多角度看待数据，用“cube”这个术语来表示多维的概念）形式表示数据的能力，这就使得数据挖掘成为可能。反过来，当数据挖掘方面的研究人员提高了实现多维数据集的技术时，这些成果也给数据库设计领域带来了好处。

最后，我们应当认识到，成功的数据挖掘远不止包括数据集范围内的模式识别。明智的判断还要确定哪些模式是有实际意义的，哪些只是偶然的。某个便利店卖出了大量彩票这样一个事实对于计划买彩票的某个人来说，不可能有什么重要意义，但是对于食品杂货店经理来说，发现有些买了快餐的顾客也常会买点冷冻食品，那就是一条很有意义的信息了。同样，数据挖掘也包括了大量的道德方面问题，包括在数据仓库中表述的个体的权利、所得结论的准确性和用处，甚至涉及数据挖掘初衷是否恰当。

问题与练习

1. 为什么数据挖掘不在“联机”数据库上实施？

2. 试举出另一个模式的例子，要求文中提到的每种数据挖掘类型都可以在此例中找到。
3. 给出几种不同的观点，可以在挖掘销售数据中用到多维数据集。
4. 数据挖掘与传统数据库查询有何不同？

9.7 数据库技术的社会影响

随着数据库技术的发展，以往不可能得到的信息现在可以获取到。许多情况下，自动化图书馆系统记录了每个用户的阅读记录，零售商保存了每个客户的购买记录，因特网搜索引擎保留了客户端的请求记录。而且，这些信息对以下群体也具有潜在的价值：市场营销公司、法律实施机构、雇主以及私有个体。

这代表了渗透到数据库应用整个范围的潜在问题。基于现在的技术，收集大量的信息、合并或比较不同的数据集合以获得它们之间的关系，变得非常容易，而这些信息以前则是不可获取的。这种衍生物（如同是一把双刃剑）非常庞大，它不仅是学术界辩论的主题，更是真实存在的事实。

现在的数据收集工作在有些情况下比较明显，而在有些情况下就显得比较微妙了。前一种情况的例子是直接要求某人提供信息。这可能以自愿的方式进行，如民意调查或竞赛登记等形式；也可能以非自愿的方式进行，如以政府规定强制进行等。有时，自愿与否取决于个人的观点。当申请借贷时提供个人信息，是自愿还是非自愿？这种不同取决于获得贷款是为了方便还是必需的。现在有些零售店使用信用卡时，要求以数字化格式记录签名。同样，提供这种信息是否自愿，也是取决于所处的环境。

数据收集比较微妙的情况下，就避免了与对象直接进行交流。这样的例子有：信用卡公司记录下了信用卡持有者的所有购物活动，网站记录下了访问者的身份，社会活动调查员记录下了停在目标单位停车场的汽车的车牌号。在这些情况下，数据收集的对象不会意识到他们的信息被收集，更不大可能知道存在为此建立的数据库。

有时候，如果停下来想想，这种潜在的数据收集活动就很清楚了。例如，杂货店可能会为已经登记过的常客提供折扣。登记过程中可能要发一个身份认证卡，在购物时要出示该卡才能享受折扣。这样商店就可以收集大量客户的购物记录，而这种记录的价值远远超出了折扣的价值。

当然，推动数据收集繁荣发展的动力就是数据的价值，它的作用因数据库技术的发展而得到扩大，这些数据库技术使数据能够联系起来，这就揭示出了原本隐藏的信息。例如，对信用卡持有者的消费模式进行分类和交叉列表，就能获得极具市场价值的顾客资料概况。利用这些信息，健美杂志就可向那些最近买过健身器材的人寄去订阅单，而驯狗杂志的订阅单则会寄给那些前不久买过狗食的人。有时候，信息的组合方式实在是富有想象力。如将犯罪记录与社会福利记录进行对比，可以找到和抓获假释期间的违法者；1984年美国的义务兵役机构利用从一家著名的冰淇淋店获得的生日登记表，找出了那些逃避兵役登记的公民。

有一些方法能够用来保护社会，防止数据库滥用。一种办法就是通过法律手段。但是，通过一个法案来反对一种行为，仅仅是让这种行为不合法，但阻止不了行为的发生。最好的例子是1974年美国通过的隐私权法案，其目的是保护公民，防止政府滥用数据库。该法案中一个条款规定政府部门要在联邦注册署公布其数据库通告，允许公民访问和纠正他们的个人信息。然而，政府部门却迟迟不能遵照这个条款。这倒并非一定说明那些部门是出于什么恶意的目的，在许多情况下，是属于官僚作风的问题。但是，官僚机构构建的人事数据库不能有效鉴别身份这样一个事实，却是令人不安的。

另一个也许更有效地控制滥用数据库的办法是公众舆论。如果损失大于好处，人们就不会去滥用数据库，而且，企业最害怕的惩罚就是负面的公众舆论，因为这将直击要害。20世纪90年代初期，正是公众舆论阻止了一些主要信贷机构为商业用途出售其邮件列表。更近一点的例子，美国在线（一家主要的因特网服务提供商）在公众压力下，放弃了向电话销售员出售客户相关信息的政策。即使是政府机构也会向公众舆论妥协。1997年，美国社会保障局修改了通过因特网查阅社会保障记录的计划，这是因为公众舆论对信息的安全性产生了质疑。在迫于公众舆论压力的情况下，几天就能得到结果，这与冗长的司法过程完全不同。

当然，在许多情况下，数据的持有者和数据的主体都受益于数据库应用，但是在所有情况下，都不能轻视隐私的丢失。当信息准确时，私密性问题就比较严重；而当信息错误时，秘密性问题就变得硕大无比。当意识到自己的信用度受到了错误信息的负面影响时，你可以想象出那种无望的感觉。不难想象，在一个错误信息很容易被传开的环境里，问题会怎样地扩大。

一般来说，秘密性问题是并且仍将是技术（特别是数据库技术）进步带来的一个主要副作用。要解决这些问题就需要我们成为有素质的、警觉的、积极的公民。

问题与练习

1. 是否能赋予执法部门为了确定有犯罪倾向的人而访问数据库的权利，即使这些人并无前科？
2. 是否能赋予保险公司为了确认有潜在健康问题的人而访问数据库的权利，即使这些人并无任何症状？
3. 假设你的经济情况良好。如果这个信息在很多机构传开，从中你会获得什么好处？同样信息的散布，又会有什么不利？又若你的经济情况不理想，结果又将如何？
4. 新闻出版自由在控制数据库的滥用上起到什么作用？（例如，新闻影响公众舆论或曝光数据库滥用到了何种程度？）

复习题

（带*的题目涉及选读小节的内容。）

1. 概述平面文件与数据库之间的不同。
2. 数据独立性是什么意思？
3. 在数据库实现的层次化方法中，DBMS的作用是什么？
4. 模式和子模式有什么不同？
5. 指出把应用软件与DBMS分离的两个好处。
6. 描述抽象数据类型（第8章中讲到的）与数据库模型的相似之处。
7. 说出下列情况或活动在数据库系统（用户、应用软件程序员、DBMS软件设计者）中发生的级别：
   a. 数据在磁盘上怎样存储效率才最高？
   b. 243航班还有空位吗？
   c. 在大容量存储器中，关系应当如何组织？
   d. 允许用户敲错几次口令才终止对话？
   e. 怎样才能实现PROJECT运算？
8. 下列哪一项工作是由DBMS完成的？
   a. 确保用户对数据库的访问权限制在相应的子模式内。
   b. 把基于数据库模型的一些指令翻译成对实际数据存储系统的活动。
   c. 隐藏数据库中的数据分散在网络中的许多计算机内这一事实。
9. 在一个关系数据库中，怎样表示以下有关航空公司、航班（对某一天而言）和乘客的信息：
   航空公司：Clear Sky、Long Hop、Tree Top
   Clear Sky的航班：CS205、CS37、CS102
   Long Hop的航班：LH67、LH89
   Tree Top的航班：TT331、TT809
   Smith已预订CS205（12B座）、CS37（18C座）和LH89（14A座）。
   Baker已预订CS37（18B座）和LH89（14B座）。
   Clark已预订LH67（5A座）和TT331（4B座）。

10. 对一个关系应用SELECT和PROJECT运算的次序有什么意义？或者说，在怎样的条件下，先做SELECT再做PROJECT的结果，与先做PROJECT运算再做SELECT操作的结果一样？
11. 给出一个论据，证明（如9.2节描述的）在JOIN运算中where子句是不必要的。（也就是说，要证明任何用到where子句的查询语句都能够通过下面这样的方式重新表示：用JOIN操作把一个关系中的每一个元组与另一个关系中的每一个元组连接起来。）
12. 对下列关系，执行以下各指令后，关系RESULT是怎样的：

X关系

U	V	W
A	Z	5
B	D	3
C	Q	5

Y关系

R	S
3	J
4	K

a. RESULT←PROJECT W from X
b. RESULT←SELECT from X where W = 5
c. RESULT←PROJECT S from Y
d. RESULT←JOIN X and Y where X.W ≥ Y.R

13. 根据以下数据库，利用SELECT、PROJECT和JOIN命令，写出指令序列来回答下列有关部件与生产商的问题：

PART 关系

PartName	Weight
Bolt 2X	1
Bolt 2Z	1.5
Nut V5	0.5

MANUFACTURER 关系

CompanyName	PartName	Cost
Company X	Bolt 2Z	.03
Company X	Nut V5	.01
Company Y	Bolt 2X	.02
Company Y	Nut V5	.01
Company Y	Bolt 2Z	.04
Company Z	Nut V5	.01

a. 哪些公司生产了Bolt 2Z？
b. 获取一个由Company X生产的部件及其价格清单。
c. 哪些公司生产重量为1的部件？

14. 用SQL回答第13题。
15. 利用SELECT、PROJECT和JOIN命令，写出指令序列来回答以下几个有关图9-5中的EMPLOYEE、JOB和ASSIGNMENT关系中信息的问题：
   a. 获取公司员工姓名和地址清单。
   b. 获取在人事部工作和曾经工作过的人员姓名和地址清单。
   c. 获取正在人事部工作的人员姓名和地址清单。
16. 用SQL回答上题。
17. 设计一个包含作曲家、生平及其作品信息的关系数据库（避免类似于图9-4中的冗余）。
18. 设计一个包含乐队、唱片以及所录乐曲的作曲者信息的关系数据库（避免类似于图9-4中的冗余）。
19. 设计一个包含计算设备生产商及其产品的关系数据库（避免类似于图9-4中的冗余）。
20. 设计一个包含有关出版商、杂志及订户信息的关系数据库（避免类似于图9-4中的冗余）。
21. 设计一个包含有关零部件、供应商及客户信息的关系数据库。每种零部件可有几个供应商供应，并可有多个客户订购。每个供应商可以供应许多种零部件，也可有多个客户。每个客户可以向多个供应商订购多种零部件；实际上，同一种零部件可向一个以上的供应商订购（避免类似于图9-4中的冗余）。
22. 写出指令序列（利用SELECT、PROJECT及JOIN运算）来实现从图9-5所示的关系数据库中检索财务部每个职务的JobId、StartDate及TermDate。
23. 用SQL回答上题。
24. 写出指令序列（利用SELECT、PROJECT及JOIN运算）来实现从图9-5所示的关系数据库中检索现任每位员工的Name、Address、JobTitle及Dept。
25. 用SQL回答上题。
26. 写出指令序列（利用SELECT、PROJECT及JOIN运算）来实现从图9-5所示的关系数据库中检索现任每个员工的Name及JobTitle。
27. 用SQL回答上题。
28. 由单一关系

Name	Department	TelephoneNumber
Jones	Sales	555-2222
Smith	Sales	555-3333
Baker	Personnel	555-4444

和两个关系

Name	Department
Jones	Sales
Smith	Sales
Baker	Personnel

Department	TelephoneNumber
Sales	555-2222
Sales	555-3333
Personnel	555-4444

提供的信息有什么不同？

29. 设计一个包含汽车部件及其子部件的关系数据库。要做到：一个部件可以包含更小的零件，同时它本身可以是更大部件的零件。
30. 选择一个常用的网站，像www.google.com、www.amazon.com或www.ebay.com，设计一个关系数据库，作为网站的支持数据库。
31. 基于图9-5所示的数据库，说明以下程序段回答的问题：

```
TEMP←SELECT from ASSIGNMENT where
  TermDate = "*"
RESULT←PROJECT JobId, StartDate
  from TEMP
```

32. 把上题中的查询翻译成SQL语句。
33. 基于图9-5所示的数据库，说明以下程序段回答的问题：

```
TEMP1←JOIN EMPLOYEE and ASSIGNMENT
  where EMPLOYEE. EmplId =
ASSIGNMENT. EmplId
TEMP2←SELECT from TEMP1
  where TermDate = "*"
RESULT←PROJECT Name, StartDate from
  TEMP2
```

34. 把上题中的查询翻译成SQL语句。
35. 基于图9-5所示的数据库，说明以下程序段回答的问题：

```
TEMP1←JOIN EMPLOYEE and JOB
   where EMPLOYEE. EmplId=JOB.EmplId
TEMP2←SELECT from TEMP1
   where Dept="SALES"
RESULT←PROJECT Name from TEMP2
```

36. 把上题中的查询翻译成SQL语句。
37. 把SQL语句

```
select JOB.JobTitle
from ASSIGNMENT,JOB
where ASSIGNMENT.JoblId = JOB.JobId
   and ASSIGNMENT.EmplId = "34Y70"
```

翻译成SELECT、PROJECT及JOIN运算的序列。

38. 把SQL语句

```
select ASSIGNMENT.StartDate
from ASSIGNMENT, EMPLOYEE
where ASSIGNMENT.EmplId =
    EMPLOYEE.EmplId
  and EMPLOYEE.Name = "Joe E. Baker"
```

翻译成SELECT、PROJECT及JOIN运算的序列。

39. 说明对第13题中数据库执行以下SQL语句后的效果：

```
insert into MANUFACTURER
values('Company Z','Bolt 2X',.03)
```

40. 说明对第13题中数据库实施以下SQL语句后的效果：

```
update MANUFACTURER
  set Cost=.03
  where CompanyName = 'Company Y'
    and PartName = 'Bolt 2X'
```

*41. 请确定用来维护杂货店库存的面向对象数据库中的几个对象，并说明每个对象中应该包含哪些方法？

*42. 请确定用来维护图书馆藏书记录的面向对象数据库中的几个对象，并说明每个对象中应该包含哪些方法？

*43. 如果T1和T2两个事务按如下安排来调度，会产生什么样的错误信息？
T1设计为计算账户A和B总和，T2设计为从账户A转账100美元到账户B。T1先读取账户A的余额，然后T2执行转账，最后T1读取账户B的余额，并输出读得的两个值的总和。

*44. 说明怎样用文中介绍的锁定协议来解决问题43。

*45. 第43题中如果T1是较新的事务，那么受伤等待协议会对上述事件序列起什么作用？如果T2是较新的事务，结果又如何？

*46. 假设有一个事务试图在一个余额为200美元的账户中存入100美元，同时另一事务试图从同一账户取出100美元。描述如何通过这些事务的交叉处理，使得最后余额为100美元。描述如何通过这些事务的交叉处理，使得最后余额为300美元。

*47. 一个事务对数据库中的一个数据项，有互斥访问和有共享访问有什么不同？为什么这种差别很重要？

*48. 9.4节讨论过的关于并发事务的一些问题不仅限于数据库环境。当用文字处理程序来访问同一个文档时会产生怎样类似的问题？（如果你的PC机中有字处理程序，试着激活两个实例来访问同一文档，看看会发生什么。）

*49. 假定一个顺序文件有50 000条记录，查询一条记录需5 ms。请问：检索一条处在文件中间部位的记录，需要等待多长时间？

*50. 见图9-15中的归并算法，如果其中一个输入文件一开始就是空的，请列出执行该算法的步骤。

*51. 修改图9-15中的算法，来处理这样一种情况：两个输入文件都包含一个有同样键字段值的记录。假定这些记录都一样，在输出文件里只要有一个即可。

*52. 设计一个系统，利用这个系统，存储在磁盘上的一个文件能够作为顺序文件以两种不同的顺序能进行处理。

*53. 说明如何能够利用一个文本文件作为基本结构，来构建一个包含杂志订阅者信息的顺序文件。

*54. 设计一种技术，通过这种技术，将逻辑记录大小并不一致的顺序文件作为文本文件来实现。例如，假设要构建一个包含小说家信息的顺序文件，其每条逻辑记录都包含一个作家的信息及其作品清单。

*55. 索引文件与散列文件相比，有什么优势？而散列文件与索引文件相比，又有什么优势？

*56. 本章描述了传统文件索引与由操作系统维护的文件目录系统相似之处。在哪些方面操作系统的文件目录与传统索引不同？

*57. 如果散列文件分到10个桶里，那么任意3条记录中的至少2条放进同一个桶的概率是多少（假定散列函数不让任何一个桶拥有优先权）？文件中必须存放有多少条记录，才可能发生碰撞？

*58. 假设文件被分进100个桶而不是10个桶，重解上题。

*59.如果我们利用本章讨论过的除法技术作为散列函数，并且将文件存储区分成23个桶，那么当把键翻译为一个二进制值时，应当搜寻哪个区来寻找其键值为整数124的记录？

*60. 通过比较散列文件的实现与同构二维数组的实现，说明散列函数与地址多项式的作用有什么类似之处？

*61. 试给出下列比较中的一个优点：
   a. 顺序文件优于索引文件。
   b. 顺序文件优于散列文件。
   c. 索引文件优于顺序文件。
   d. 索引文件优于散列文件。
   e. 散列文件优于顺序文件。
   f. 散列文件优于索引文件。

*62. 从哪些方面可以看出顺序文件类似于链表？

社会问题

下面的问题有助于你分析一些与计算领域相关的道德、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 在美国，所有囚犯的DNA记录都存储在一个数据库中，以备犯罪研究所用。如果发布这些信息用于其他用途，例如用作医学研究，这样做道德吗？如果合乎道德，可用于什么目的？如果不合乎道德，为什么？而每种情况的利与弊又是什么？
2. 大学能将其学生的信息公开到何种程度？可以公布他们的姓名和地址吗？可以在学生不知情的情况下公布他们的成绩排名吗？你的看法是否与第1题的答案一致？
3. 构建有关个人的数据库时，采取什么样的限制比较合适？政府有权掌握公民的什么信息？保险公司有权掌握其客户的什么信息？公司有权掌握其雇员的什么信息？在这些情况中，需要实行控制吗？如果需要，怎样实现？

4. 如果信用卡公司把它的客户的消费模式卖给商业公司，这样做是否合适？如果赛车邮购业务公司把它的邮购清单卖给赛车杂志，这样做是否合适？如果美国国税局把那些有着巨额收入的纳税人的姓名和地址信息卖给股票经纪人，这样做是否合适？如果你没有充分的把握回答是与否，那么你有什么可行的方案？
5. 数据库的设计者对于如何使用数据库信息应当负怎样的责任？
6. 假设数据库的信息因数据库错误而被非授权用户访问。如果信息被怀有恶意目的的用户获得和使用，那么数据库设计者应对此承担何种责任？你的回答是否与作恶者为发现数据库设计漏洞并非法获取信息所花费的精力大小有关？
7. 数据挖掘的盛行带来了大量的道德和隐私问题。如果数据挖掘揭示了你所在社区的所有居民的某些特性，那么你的隐私是否受到侵犯？数据挖掘的使用是促进了商业的发展还是鼓励了盲从？因为相对于个别问卷调查明确询问的方式而言，从人口普查的数据中能提取更多的信息，那么强制公民参加人口普查是否合适呢？数据挖掘给予商业公司的好处，对于不知情的客户来说是否不公平？这样一种状况的好与坏，到了何种程度？
8. 可以允许公司或个人收集和保留私人信息能够到多大的程度？尽管收集的信息分散在一些发起者之间，但是如果这些信息已经能公开地获得，那么现在该怎么办？公司或个人期望在何种程度上保护这类信息？
9. 许多图书馆提供参考查询服务，所以读者在查阅信息时可以得到图书管理员的帮助。因特网和数据库技术的出现是否会使这种服务过时？如果会，那么这是前进了还是倒退了？如果不会，为什么？因特网和数据库技术的存在对图书管理员本身有什么样的影响？
10. 你的身份信息被盗用的可能性到了什么程度？你会采取哪些步骤使盗用机会最小？如果你的个人信息被窃，对你的伤害将有多大？如果发生这种情况，你自己有责任吗？

课外阅读

Beg, C. E. and T. Connolly. *Database Systems: A Practical Approach to Design, Implementation and Management*, 4th ed. Boston, MA: Addison-Wesley, 2005.

Berstein, A , M. Kifer and P. M. Lewis. *Database Systems*, 2nd ed. Boston, MA: Addison-Wesley, 2006.

Date, C. J. *An Introduction to Database Systems*, 8th ed. Boston, MA: Addison-Wesley, 2004.

Date, C. J. *Databases, Types and the Relational Model*, 3rd ed. Boston, MA: Addison-Wesley, 2007.

Elmasri, R. and S. Navathe. *Fundamentals of Database Systems*, 6th ed. Boston,MA : Addison-Wesley, 2011.

Patrick, J. J. *SQL Fundamentals*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

Silberschatz, A. H. Korth, and S. Sudarshan. *Database Systems Concepts*，8th ed. New York: McGraw-Hill, 2009.

Ullman, J. D. and J. D. Widom. *A First Course in Database Systems*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2008.

第10章

计算机图形学

本章将探索计算机图形学领域，这是一个对电影制作和交互式视频游戏具有重大影响的领域。实际上，计算机图形学的发展解除了视觉媒体对实体的限制，许多人认为计算机动画在不久的将来会取代整个影视产业对传统的演员、布景和照片的需求。

本章内容

计算机图形学是计算机科学的分支，它应用计算机技术创建和操控视觉表现。这是一个广泛的主题，它包括：文本表示、图形和图表的创建、图形化用户界面的开发、照片的操作、视频游戏的制作、影视动画的生成等。然而，术语计算机图形学越来越多地被用来指代3D图形学的特定领域，本章大部分内容将集中在这个主题上。我们将从定义3D图形学开始，阐明它在广义的计算机图形学中的作用。

10.1 计算机图形学的范围

随着数码相机的出现，数字图像处理软件迅速流行起来。人们可以使用这类软件通过去除污点和“红眼”等操作达到“润色”照片的目的，也可以在不同的照片中进行裁剪和粘贴，创建一幅并非反映真实世界的图像。

类似的技术经常应用于电影和电视产业中，以产生特效。例如，可以很容易地通过去除支撑的金属丝、重叠多幅图像，或产生新的图像序列帧等特效处理，改变最初拍摄的情节。这促使影视产业摒弃像胶卷之类的模拟系统，转向数字编码的图像。

除了处理数字照片和视频的软件外，现在还有各种各样的工具/应用软件包，它们帮助产生二维图像，从简单的画线到复杂的艺术品。（一个众所周知的最基本的例子就是微软的“画图”应用程序。）这类程序的基本操作包括：绘制点和线、插入椭圆和矩形这类简单的几何图形、给区域填充颜色，以及裁剪和粘贴绘图的指定部分。

注意，上面所有的应用都是处理平面二维图形和图像的操作。这里有两个相关研究领域，一个是**2D图形学**（2D graphics），另一个是**图像处理**（image processing）。二者的区别在于：2D图形学侧重于把二维图形（圆、矩形、文字等）转化为像素模式，产生图像；而图像处理侧重于分析图像中的像素，进行模式识别，以达到增强或“理解”图像的目的。简言之，2D图形学

处理生成图像，而图像处理分析图像。

与2D图形学中把二维图形转化为图像相对应，**3D图形学**（3D graphics）领域处理的是把三维图形转化成图像。这个过程是：建造三维场景的数字化版本，然后模拟照相的过程，产生这些场景的图像。这与传统的摄影类似，不同之处在于场景是使用3D图形技术“拍摄”出来的，实际是不存在的，存在的是数据和算法的集合。因此，3D图形“拍摄”虚拟世界（如图10-1所示），而传统的摄影技术拍摄真实世界。

需要注意的是，使用3D图形创建图像要经历两个不同的步骤：一个是创建、编码、存储以及操作被拍摄出来的场景；另一个是生成图像的过程。前者是创造性的、艺术的过程，而后者则是以计算为主的过程。这些主题是我们在下面4个小节中将要讨论的。



图10-1　使用3D图形产生的虚拟世界的“照片”（迪士尼与皮克斯合拍的《玩具总动员》剧照）© Disney/Pixar

3D图形可以制作出不依赖于实体的虚拟场景，这使得它非常适用于交互式视频游戏和动画电影的制作。交互式视频游戏由数字化的三维虚拟环境构成，游戏玩家与之进行交互，玩家看到的图像是通过3D图形技术制作出来的。动画电影是用类似的方法创建的，不同之处在于只是动画制作者与虚拟环境交互，而公众看到的则是导演/制片人发布的二维图像帧序列。

本书将在10.6节中更全面地讨论3D图形学在动画中的应用。这里可以想象一下，随着3D图形技术的发展，这些应用将可能导向何处。如今，电影是作为二维图像序列发布的。尽管显示这些信息的放映机已经取得了很大进步，从使用胶卷的模拟设备到使用DVD播放机和平板显示器的数字技术，但它们的显示仍然只是二维的。

但是，想象一下当创建和操作真实的三维虚拟世界的能力得到改善时，将会发生什么改变。我们将不再仅能“拍摄”这些虚拟世界和以二维图像的形式发布电影，而且能发布虚拟世界。观众将不仅仅只能观看电影，还可以通过“3D图形放映机”来观看虚拟场景，就像通过专用的“游戏盒”来观看视频游戏一样。观众可能先看到导演/制片人预定的“推荐情节”，与此同时还可以与虚拟场景交互，就像玩视频游戏一样产生另外的场景。考虑到正在研发的三维人机接口的巨大潜力，未来的前景十分诱人。

问题与练习

1. 总结图像处理、2D图形学和3D图形学之间的区别。
2. 3D图形学与传统摄影有何不同？
3. 应用3D图形学制作“照片”的两个主要步骤是什么？

10.2 3D图形概述

本章我们从创建和显示图像的整个过程来开始对3D图形学的研究。这个过程由3步构成：建模、渲染（rendering）和显示。建模（将在10.3节中详细介绍）与传统电影产业中设计和构造一个场景类似，不同之处在于3D图形场景是用数据结构和算法"构造"的。这就导致应用计算机图形学产生的场景可能在现实中永远都不存在。

下一步就是通过计算场景中的物体如何显示在由特定位置的相机拍摄的照片中，来生成场景的二维图像。这称为**渲染**（rendering，10.4节和10.5节的主题），渲染的概念是运用解析几何，来计算场景中的物体到一个称为**投影平面**（projection plane）的面上会形成的投影，这种方式与相机将场景投影到胶卷上的方式类似（如图10-2所示）。这种投影称为**透视投影**（perspective projection），在这种投影方式下，所有的目标都沿着一条称为**投影线**（projector）的直线向前延伸，这条直线是从一个称为**投影中心**（center of projection）或**视点**（view point）的公共点延伸出来的。这与**平行投影**（parallel projection）不同，顾名思义，平行投影线是平行的。透视投影产生的投影类似于人类眼睛所看到的，而平行投影产生的是物体"真正"的剖面，这在工程绘图中非常有用。

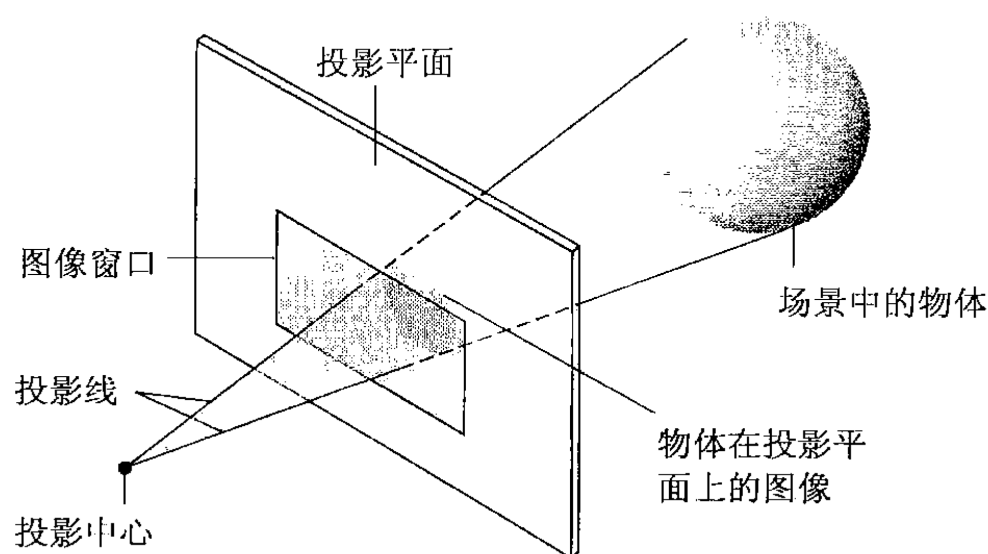


图10-2 3D图形学范例

对于用来定义最终图像边界的投影平面，其中受限的部分称为**图像窗口**（image window）。它对应于显示在大多数相机取景器上的矩形，指明潜在图像的边界。实际上，大多数相机的取景器允许用户看到相机投影平面上更大的区域，而不仅仅是图像窗口。（你可能会在取景器中看到"玛莎阿姨"头部的上方，但是，除非这部分影像也出现在图像窗口中，否则它就不会出现在最终图像中。）

投影到图像窗口的场景部分确定了以后，就可以计算出最终图像上的每个像素点的显示情况，这种逐个像素的计算过程可能会很复杂，因为它需要确定场景中的物体如何与光线融合。（在明亮光线下硬且有光泽的表面与在间接光线下软且透明的表面，二者的渲染方法应该有所不同。）因此，渲染处理涉及包括材料科学和物理学在内的许多其他研究领域。而且，在决定一个物体的显示效果时经常需要了解场景中的其他物体。这个物体可能处在另一个物体的阴影中，或者这个物体是镜子，它的外观实质上就是其他物体。

当确定了每个像素的外观后，结果被集中地表示成图像的位图，并存储在称为**帧缓冲区**（frame buffer）的存储区域中。这个缓冲区可能是主存中的一个区域，或当有专门处理图形应用的硬件时，它可能是专用存储电路中的一个块。

最后，存储在帧缓冲区的图像或者为了观看而显示，或者为以后的显示而传送给更永久的

存储器。如果生成的图像将用于电影画面，那它可能在最终显示前被存储甚至是被修改。但是，在交互式视频游戏或飞行模拟器中，图像必须显示，因为它们是在实时生成的，这个要求经常限制了图像的质量。这就是由制片厂发布的功能齐全的动画产品的图像质量要超过当今交互式视频游戏中的图像质量的原因。

最后，我们通过分析一个典型的视频游戏系统来结束对3D图形的介绍。这个游戏实际上就是一个编码的虚拟世界和软件的结合体，它允许游戏玩家操控这个虚拟世界。当玩家操控这个世界时，游戏系统会不断地渲染场景并把图像存储到图像缓冲区中。为了克服真实世界的时间限制，大多数渲染处理都是由专用硬件来实现的。实际上，正是这些硬件使游戏系统和一般个人计算机之间有了显著差别。最后，游戏系统中的显示设备显示了帧缓冲区中的内容，给玩家以变化场景的幻觉。

问题与练习

1. 总结在使用3D图形生成图像时涉及的3个步骤。
2. 投影平面和图像窗口之间有何不同？
3. 什么是帧缓冲区？

10.3　建模

3D计算机图形投影的起始阶段与戏剧舞台制作方式十分相似：必须设计出布景，收集或者搭建所需的道具。在计算机图形学的术语中，布景称为**场景**（scene），道具称为**物体**（object）。记住，3D图形场景是虚拟的，因为组成它的物体是由数字化编码模型“构建”而成，并不是实际的物理结构。

本节将探讨与“构建”物体和场景有关的话题。我们以单个物体的建模问题开始，并以考虑收集这些物体以形成场景这个任务结束。

10.3.1　单个物体的建模

在舞台制作中，道具的真实程度取决于它在场景中的使用方式。我们可能不需要一辆完整的汽车，电话并不需要能用，背景可能也只是画在大背景屏幕上的。同样，就计算机图形学而言，一个物体的软件模型能准确地反映物体真实属性的程度依赖于情境的需要。前景物体的建模与背景中的物体相比需要考虑更多的细节。而且，在那些没有严格实时限制的情况下，会产生更多的细节。

因此，一些物体模型可能相对简单，而另一些可能极其复杂。作为一个通用规则，模型越精确，图像的质量越高，渲染所需要的时间也就越长。因此，现在进行的大多数对于计算机图形学的研究都是在寻求一种开发技术，以构建非常精细，同时又不失高效的物体模型。这些研究中有些涉及开发模型，开发模型依据物体在场景中的最终作用来提供不同的细节层次，这样可以在变化的场景中重用同一个物体模型。

描述一个物体所需的信息包括：物体的形状，以及额外的特性（如决定物体如何与光线交互的表面特性等）。现在，让我们考虑形状建模这个任务。

1. 形状

在3D图形中物体的形状通常描述成称为**平面片**（planar patch）的小平面的集合，其中每一个都是一个多边形。这些多边形形成了**多边形网格**（polygonal mesh），它近似于被描述的物体

的形状（如图10-3所示）。通过使用小平面片，近似值可达到所需的精度。

多边形网格中的平面片经常选择三角形，因为每个三角形能用它的3个顶点来表示，这是在三维空间中确定一个平面所需点的最少数目。在任何情况下，多边形网格都表示成这些平面片顶点的集合。

一个物体的多边形网格表示可以通过多种途径获得。其中一种是：以所需形状的精确的几何描述开始，然后用这些描述构建多边形网格。例如，解析几何中半径为r的球（中心在原点）用方程来描述：

$$r^2=x^2+y^2+z^2$$

基于这个公式，我们可以建立球上经线和纬线的方程，标识这些线的交叉点，然后使用这些点作为多边形网格中的顶点。类似的技术可以应用到其他传统的几何形状上，这就是为何在廉价的计算机动画中人物角色经常甲球、圆柱体和锥体这些结构拼凑的原因。

更一般的形状可以用更复杂的分析方法来描述。其中一种方法是使用**贝塞尔曲线**（Bezier curve）（以皮尔·贝塞尔命名，他在19世纪70年代早期提出了这个概念，当时他是雷诺汽车公司的工程师），它允许在三维空间中只用几个称为控制点的点来定义曲线段（其中有两个点表示曲线段的端点，而其他的点则指出曲线的弯曲方式）。例如，图10-4显示了由4个控制点定义的曲线。注意，曲线显示为弯向两个不为端点的控制点。通过移动这些点，曲线可以被扭曲成不同的形状。（你可能曾经用过像微软的画图软件这样的绘图软件包构建曲线。）尽管我们在这里不再继续探讨这个话题，但描述曲线的贝塞尔技术可以扩展为描述三维曲面，称为**贝塞尔曲面**（Bezier surface）。因此，对于复杂表面，在获得多边形网格的过程中，贝塞尔曲面被证明是高效的第一步。

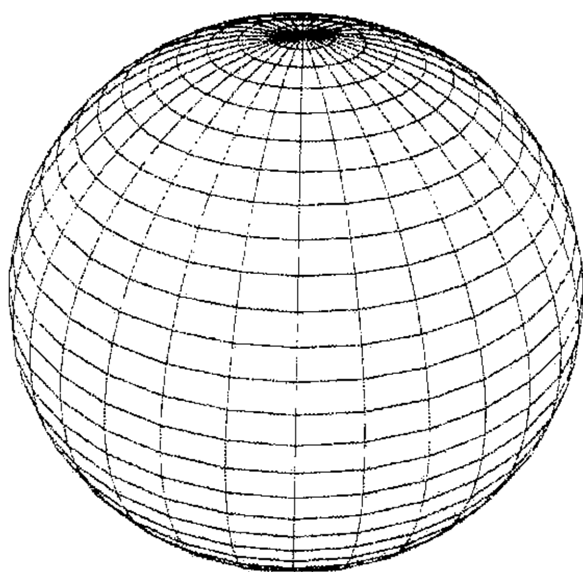


图10-3 球的多边形网格

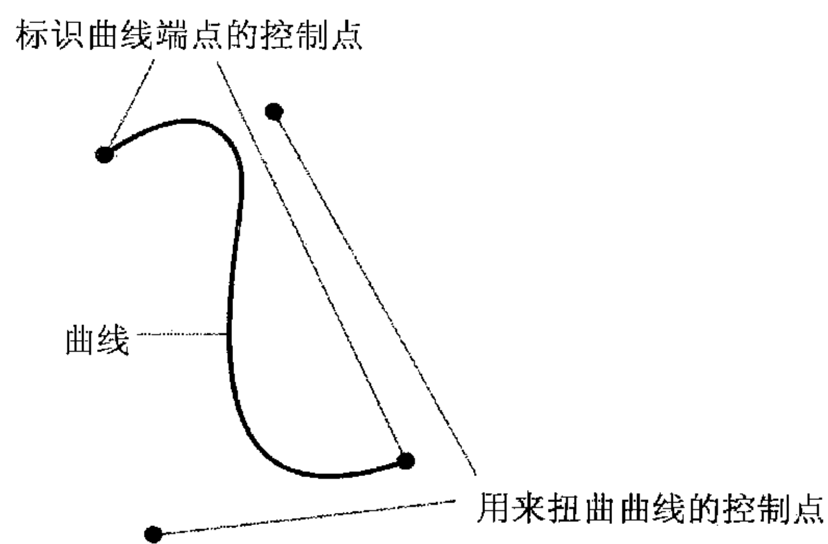


图10-4 贝塞尔曲线

你可能会问为什么需要把形状的精确描述（如球的简明公式，或描述贝塞尔表面的公式）转化为使用多边形网格的近似描述。答案是用多边形网格表示所有物体的形状确立了渲染处理的统一方法——可以更高效地渲染整个场景的技巧。这样，尽管几何公式提供了形状的精确描述，但它们只是作为构建多边形网格的工具。

另外一种获得多边形网格的方法是用蛮力的方式构建网格。在无法用优雅的数学技术表示形状的情况下，这种方法就比较常见了。在这个过程中，首先构建物体的物理模型，然后用类笔设备触摸表面，记录下模型表面点的位置，这种笔设备能记录它在三维空间中的位置，此过程就称为**数字化**（digitizing）。然后将获得的点的集合用作顶点，从而获得所描述形状的多边形网格。

遗憾的是，有些形状非常复杂，难以用几何建模或手工数字化获得真实模型。这些例子包括：复杂植物结构（比如树）、复杂地形（比如山脉），以及云、烟、火苗等气态物质等。在这些情况下，多边形网格可以通过编写自动构建所需形状的程序来获得。这样的程序统称为**程序化模型**

(procedural model)。换言之，程序化模型是应用算法产生所需结构的程序单元。

例如，通过执行下列步骤，程序化模型被用来产生山脉：以一个三角形开始，标识三条边的中点（如图10-5a所示）；然后连结这些中点，形成4个较小的三角形（如图10-5b所示）；现在在把原三角形的3个顶点固定住的同时，在三维空间里移动中间点（允许三角形的边线延长或缩短），扭曲三角形的形状（如图10-5c所示）；对于每个较小的三角形重复这个过程（如图10-5d所示），继续重复这个过程，直到达到所需的精度。

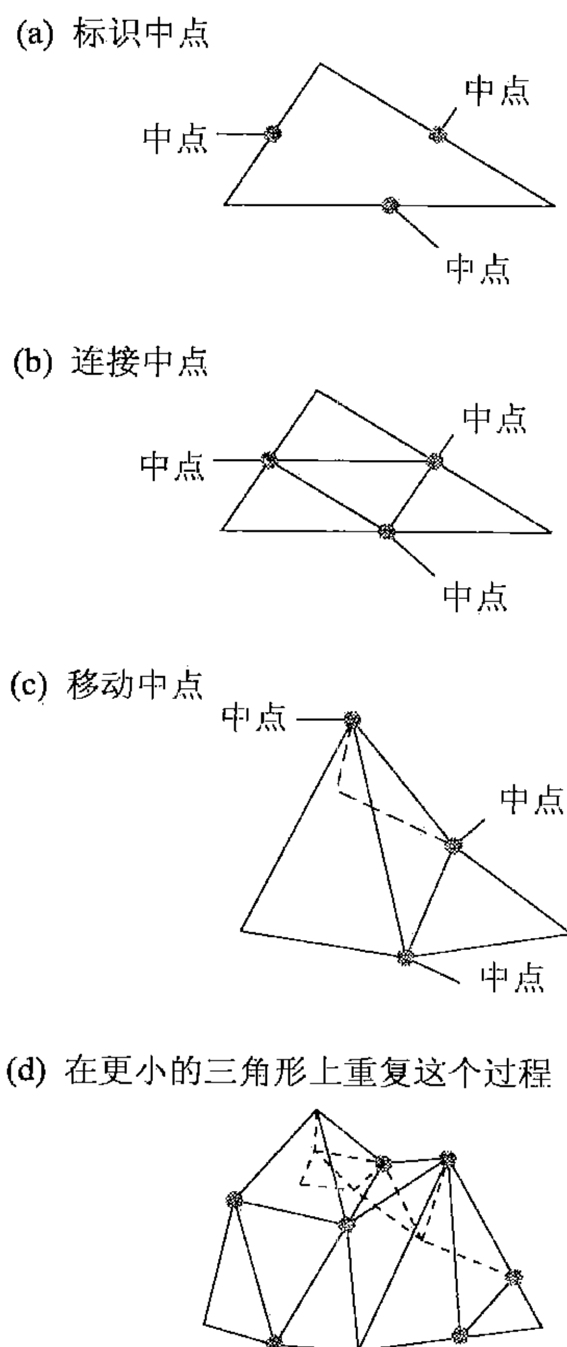


图10-5　产生--个山脉的多边形网格

程序化模型提供了一种有效的方法，来产生多个相似而又唯一的物体。例如，一个程序化模型可以用来构建各种各样逼真的树。（虽然相似，但每棵树都有自己的分支结构。）构建这些树模型的一种方法是应用分支规则，即用与词法分析器（参见6.4节）按语法规则构建词法分析树非常相似的方法来“生成”树。事实上，在这些情况下使用的分支规则的集合经常被称为语法。一个语法可能被用来“生成”松树，而另外一个可能用来“生成”橡树。

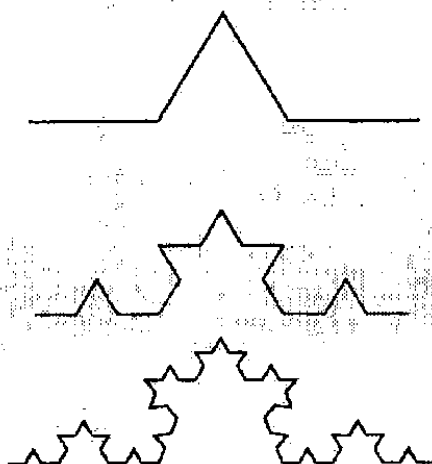
另一种构建程序化模型的方法是将物体的基础结构模拟为一个大的粒子集合。这种模型称为**粒子系统**（particle system）。粒子系统通常会应用某些预定义的规则去移动系统中的粒子（或许所用的方式会让人想起分子的交互），来生成所需的形状。例如，粒子系统已经被用来生成水面晃动的动画，我们将在后面介绍动画时看到。（想象一下，把一桶水建模为一桶玻璃弹珠，当桶滚动时，玻璃弹珠也随之四处翻滚，模拟水的运动。）粒子系统应用的其他例子包括：火苗的闪烁、云、拥挤的人群场景等。

分 形

在上文中描述了用程序化模型构建山脉（参见图10-5），这是分形作用于3D图形的例子。从技术上讲，**分形**（fractal）是“Hausdorff维度大于其拓扑维度”的几何物体。直观上讲，这意味着物体是通过低维度物体的副本“打包”而形成的。（想象一下宽度就是通过“打包”多条平行线段而创建的。）分形通常是使用递归过程来形成的，而在递归中的每个处理就是重复“打包”另外用来建立分形模式（更小）的副本。分形的结果是其每个部分都是自相似的，当放大时，它显示为自身的副本。

分形的一个传统的示例就是科赫雪花，它是通过重复地用相同结构的较小版本替换结构中的直线段而形成的。

生成的细化序列如下所示：



分形在3D图形领域经常是程序化模型的主干。实际上，它们已经被用来生成逼真的山脉、蔬菜、云和烟的图像。

程序化模型的输出通常是近似于物体形状的多边形网格。在某些情况下，如使用三角形生成山脉，网格就是生成过程的自然结果。在另外一些情况下，如应用分支规则生成树，网格可能就是额外的、最终的步骤。例如，在粒子系统中，系统外边沿上的粒子自然会被选作最终多边形网格中的顶点。

由程序化模型生成的网格的精度视具体情况而定。在场景中，用于背景中的树的程序化模型可能只要产生一个反映树基本形状的粗糙的网格，而前景中的树的程序化模型就要产生能分清各个枝叶的网格。

2. 表面特征

仅由多边形网格构成的模型只捕获了物体的形状。大多数渲染系统能在渲染过程中丰富这些模型，根据用户的需求模拟各种表面特征。例如，通过使用不同的着色技术（我们将在10.4节介绍），用户可以指定球的多边形网格被渲染成光滑的红球或是粗糙的绿球。在某些情况下，这种灵活性是可以做到的。但在需要如实渲染原始物体的情况下，关于物体的更具体的信息必须包含在模型中，这样渲染系统才会知道该干什么。

除了形状之外，还有多种有关物体信息的数字化技术。例如，沿着多边形网格的每个顶点，人们可以在物体的这一点上指定原始物体的颜色。然后在渲染过程中用这些信息重新创建原始物体的外观。

在其他的例子中，通过称为**纹理映射**（texture mapping）的处理，颜色模式能与物体表面相关联。纹理映射类似于贴墙纸，将一个预定义的图像与物体的表面相关联。这个图像可能是数字照片、艺术家的绘画，也可能是计算机生成的图像。传统的纹理图像包括砖墙、有木纹的表面和大理石表面等。

例如，假设我们需要对石墙建模，我们可以用描述长矩形体的简单多边形网格来表示墙的形状。利用这些网格，我们就能提供砖石结构的二维图像。随后，在渲染过程中，将这个图像

映射到矩形体上，产生石墙的外观。更准确地，每当渲染处理需要显示墙上的点时，它就只需显示砖石结构图像中对应的点。

当应用于相对平坦的表面时，纹理映射的效果最好。如果必须大幅扭曲纹理图像去覆盖弯曲的表面（想象成试图给一个沙滩球贴墙纸的问题），或者如果纹理图像完全裹着一个物体，并导致了接缝，在接缝处纹理模式可能不与它本身融合，那么效果看上去会不够逼真。不过，纹理映射已经被证明是一种模拟纹理的有效方法，它被广泛地用在需要实时显示的场合（一个基本的例子就是交互式视频游戏）。

3. 寻求逼真的效果

构建可以产生逼真图像的物体模型是一个正在研究的主题，特别有趣的是当前角色的材质，如皮肤、头发、毛皮和羽毛等。这些研究大多是针对特殊物质的，包括建模和渲染技术。例如，为了获得人类皮肤的逼真模型，有些学者研究光渗透到表皮和真皮层的程度以及这些层的厚度对皮肤外观的影响。

另一个例子是人类头发的建模。如果从远距离来看头发，那么传统的建模技术就足够了。但是，从近距离看，头发的显示将会是一个挑战。其中的问题包括：半透明的特性、纹理的深度、遮盖特性和头发响应像风这样的外力的方式。为了解决这些棘手的问题，有些应用程序转向对单根头发建模。（这是一个可怕的任务，因为人的头发根数的数量级达到了100 000。）但是，更令人惊奇的是有些研究者已经建立了头发模型，这个模型给出了单根头发的鳞状纹理、颜色变化和机械动力学特征等。

另外一个已经发展到相当精确建模程度的例子是布的建模。在这个例子中，利用编织模式的复杂细节，来生成织物类型（像斜纹布与缎布）之间恰当的纹理差别。将纱的细节特性与编织模式数据组合在一起，创建出编织物的模型，产生逼真的特效图像。例如，注意在图10-6中显示的史莱克袖子上的细节（还有被史莱克抱在手中的魔法猫所戴帽子上逼真的羽毛）。另外，还将物理和机械工程的知识应用到计算织物材料图像的单根线上去，以说明线的拉伸和织物修剪方面的特性。



图10-6　由梦工厂SKG制作的《怪物史莱克2》中的场景
（© Dreamworks/Picture Desk Inc./ Kobal Collection）

正如我们所说，生成逼真图像是一个活跃的研究领域，它综合了建模和渲染处理中的技术。一般来说，当取得进步时，新技术会首先应用在那些不受实时限制影响的程序中，如电影制片厂中的图形软件，建模/渲染处理与最终的图像显示之间有着明显的延迟。这些进步可以通过仔细比较迪士尼的影片《玩具总动员2》（1999年）中的角色与最初的《玩具总动员》（1995年）中

的人物而观察到。新近开发的技术被用来改善表示面部特征的多边形网格间的接缝，如鼻子与脸的其他部分的边界。当这些新的技术得到进一步发展并变得更高效时，它们就可以应用在实时系统中了，在这些环境中的图形质量也得到了改善。真正实现与虚拟世界的实时交互可能已经不太遥远了。

10.3.2 整个场景的建模

场景中的物体已经得到充分的描述和数字化编码，它们就都被赋予了场景内的位置、大小和方向。将这些信息集合并链接起来以形成一个数据结构，称为**场景图**（scene graph）。此外，场景图还包含与表示光源及相机的特殊物体的链接，其中记录了相机的位置、方向和焦点等特性。

因此，生成场景图类似于在传统的工作室中摄影。它包括布置相机、灯光、道具和背景。（当按快门时，所有的东西都对照片的外观产生了影响。）所不同的是传统照片设置包含物理实体，而场景图包含的是物体的数字化编码表示。简而言之，场景图描述的是一个虚拟的世界。

为了强调场景图的范围，再次考虑图10-1中的图像，想象一下用来生成它的场景图。人物、墙、床单、床柱、巴斯光年（太空突击队员）身后的背包、窗子的线脚、窗外的树和光源都以各自适当的精度得到了建模，并表示在场景图中。事实上，最初你可能把物体看成单个的结构，如Woody（牛仔玩偶），但实际上它们在场景图中是聚集在一起的。

场景中相机的位置会对图像产生很大的影响。正如先前提到的，物体建模的精度依赖于物体在场景中的位置。前景物体比背景物体需要更精确的建模，前景和背景的区别依赖于相机的位置。如果使用的场景环境类似于戏剧舞台布景，那么前景和背景就很好区分，物体模型也能被相应地构建。但是，如果环境要求对于不同的图像相机的位置是改变的，那么由物体模型提供的精度就需要在“照片”间进行调整，这是当前研究的一个领域。一种方法是想象场景是由“智能”模型构成，当相机在场景中移动时，这些模型重新修改了它们的多边形网格和其他特性。

移动相机情景的一个有趣的例子发生在虚拟现实系统中，用户可以借助它来体验在虚拟的三维世界里走来走去的感觉。虚拟的世界用场景图表示，而人通过操控相机来观察其中的场景。实际上，为了提供三维的深度感，可以使用两个相机：一个表示人的右眼，另一个表示人的左眼。通过显示由每只眼睛前的相机获得的图像，人们产生了居住在三维场景中的幻觉。当在体验中增加声音和触觉时，这种幻觉就变得十分逼真。

最后，我们应该注意到场景图的构建在3D图形处理中非常重要。因为它包含了生成最终图像所需的所有信息，它的完成标志着艺术建模的终止和以计算为主的图形渲染的开始。实际上，建立场景图以后，图形学的任务就变成了计算投影、确定特定点的表面精度和模拟光效。这些任务在很大程度上与特定的应用无关。

3D电视

现在有一些技术能够在电视中生成3D影像，但都依赖于同一种立体的视觉效果，即对于左右眼看到的两个稍稍不同的图像，大脑可以依此判断出深度。就这一方面而言，现今最便宜的装置需要具有滤光镜。较老的有色眼镜（20世纪50年代用于电影院中）或更现代的偏振光眼镜会过滤掉屏幕上同一图像的不同方面，从而使左右眼看到不同的图像。较昂贵的技术则需要有主动式快门3D眼镜，它与快速切换左右图像的3D电视同步地交替开关左右镜片。最后，不需要特殊眼镜或头部装置的3D电视正在研发之中，它们在屏幕表面精心排放一组滤光镜或放大镜，从而将左右图像以稍稍不同的角度投射给观看者，从而使观看者的左右眼看到不同的图像。

问题与练习

1. 下面是4个点（使用传统的直角坐标系编码），它们表示平面片的顶点。描述面片的形状。（对于没有解析几何背景知识的人，每个三元组表示从房间中的一个角落如何到达问题中的这些点。第一个数表示沿着地板和位于你右边的墙之间的接缝走多远；第二个数表示沿与位于你左边的墙平行的方向上向房间里走多远；第三个数表示从地板向上爬多高。如果有一个数是负数，你将不得不扮成一个幽灵，可以穿过墙和地板。）
   (0,0,0)　(0,1,1)　(0,2,1)　(0,1,0)
2. 什么是程序化模型？
3. 列出一些在生成一个公园图像的场景图中可能出现的物体。
4. 既然可以用几何方程更精确地表示形状，为什么还要用多边形网格来表示？
5. 什么是纹理映射？

10.4 渲染

现在让我们考虑渲染处理，它决定了当场景图中的物体投影到投影平面时，将如何显示。有几种方法可以完成渲染任务。这一节着重介绍当今“消费者市场”上流行的图形系统（视频游戏、家庭计算机等）所使用的传统方法。下一节将讨论其他两个可供选择的解决方案。

首先探讨光和物体间交互的一些背景信息。毕竟，物体的外观是由从物体发出的光决定的，因此确定物体的外观这一任务最终变成了对光的特性的模拟。

10.4.1 光－表面交互

依赖于物体的材料特性，照射到其表面的光可能被吸收，从表面反弹成反射光，或穿过表面（被弯曲）成折射光。

1. 反射

让我们考虑从一个平坦不透明表面反射的光线。光线沿直线传播，以一个角度照射到表面上，这个角度称为**入射角**（incidence angle）。光线的反射角与入射角相同，如图10-7所示。这些角度是相对于垂直于表面的线（即**法线**（normal））来测量的。（垂直于表面的线经常简单地表示为“法线”，这样就可以说“入射角是相对于法线度量的”。）入射光线、反射光线和法线在同一个平面中。

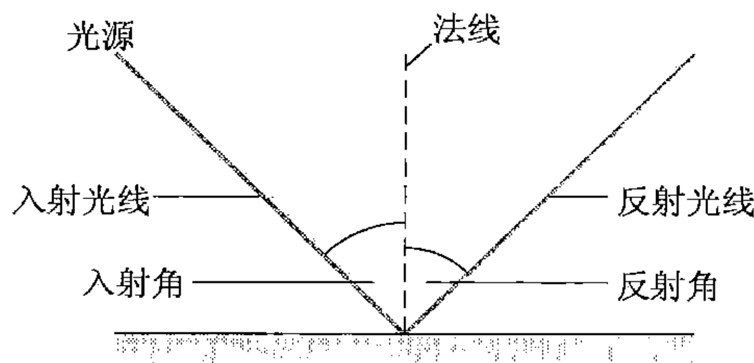


图10-7　反射光

如果表面是光滑的，在相同区域照射到表面的平行光线（如那些来自同一光源的光线）就会以相同的方向反射，并作为平行光线离开物体。这些反射光称为**镜面反射光**（specular light）。

注意，只有当表面和光源使得光反射到观察者的方向上，他才能观察到镜面反射光。它通常显示为表面上明亮的高亮区。而且，因为镜面反射光与表面的接触时间最短，这使它非常接近原始光源的颜色。

但是，物体表面很少是绝对光滑的，因此许多光线在表面照射点的方向会与大多数普通表面的不同。而且，光线经常穿透表面邻接的边界，在表面的粒子间跳弹，最后作为反射光线离开。结果是许多光线将向不同的方向散开。这种散开的光称为**散射光**（diffuse light）。与镜面反射光不同，散射光在一定范围的方向内是可见的。此外，散射光与表面的接触时间长，更容易受材料吸收特性的影响，因此它更接近物体的颜色。

图10-8表示了一个被单个光源照射的球，球上明亮的高亮区是镜面反射光产生的，通过散射光，可以看见面向光源的半球的其他部分。注意，球面背着主光源的半球通过从光源直接被反射的光是不可见的。球的这部分能够被看见是由于**环境光**（ambient light）的存在，它是“漂泊”或散开的光，不与任何特定的光源或方向相关联。被环境光照射的表面部分经常显示为统一的深色。

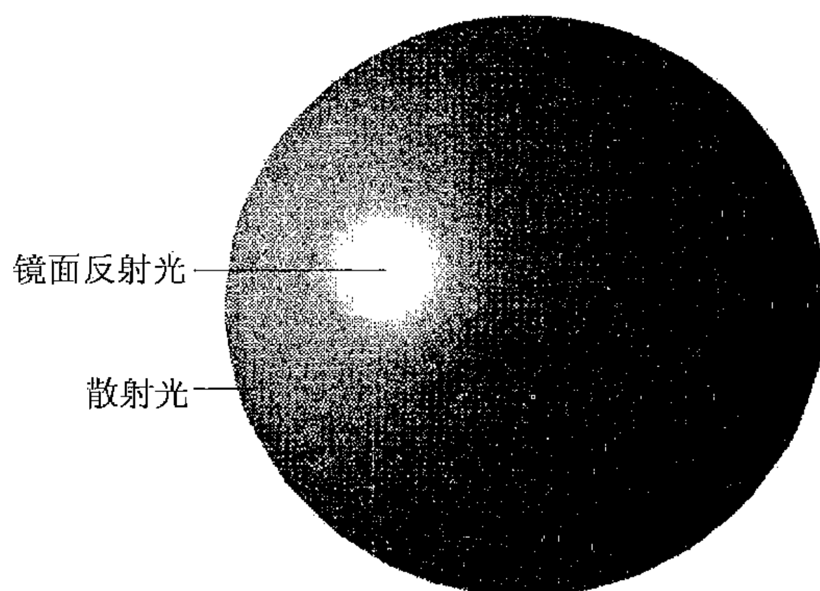


图10-8 镜面反射光与散射光

大多数物体表面既反射镜面反射光又反射散射光，表面的特征决定了镜面反射光和散射光的比例。平滑表面看起来发亮的原因在于，它们反射的镜面反射光多于散射光；粗糙表面看起来为发暗是因为它们反射的散射光多于镜面反射光。而且，由于某些表面具有细微的特性，入射光的方向不同，镜面反射光和散射光的比例也会有所不同，从一个方向入射的光照射到这样的表面上可能反射的主要是镜面反射光，而从另一个方向入射的光照射到这个表面上反射的主要是散射光。因此，当它旋转时，表面的外观将从明亮变化为灰暗。这种表面称为**各向异性表面**（anisotropic surface），不同于**各向同性表面**（isotropic surface），后者的反射模式是对称的。各向异性表面的例子可以在织物（如缎子）中找到，其中织物的细毛根据它的朝向改变材料的外观。另外一个例子是运动场的草地表面，那里草的排列方向（通常是由草被裁剪的方式决定的）产生了各向异性视觉效果，如同明暗相间的条纹图案。

2. 折射

现在考虑这种情况：光照射到透明的物体上，而非遮光的物体上。在此种情况下，光线是穿过物体而并非从其表面反射出去。当光线穿透表面时，它们的方向改变了，这种现象称为**折射**（refraction），如图10-9所示。折射程度是由相关材料的折射率决定的。折射率与材料的密度有关。高密度材料往往比低密度材料具有更高的折射率。当光线进入折射率更高的材料中时（如从空气进入水中），它在入射点处向靠近法线的方向弯曲；如果光线进入到折射率较低的材料中，它将向远离法线的方向弯曲。

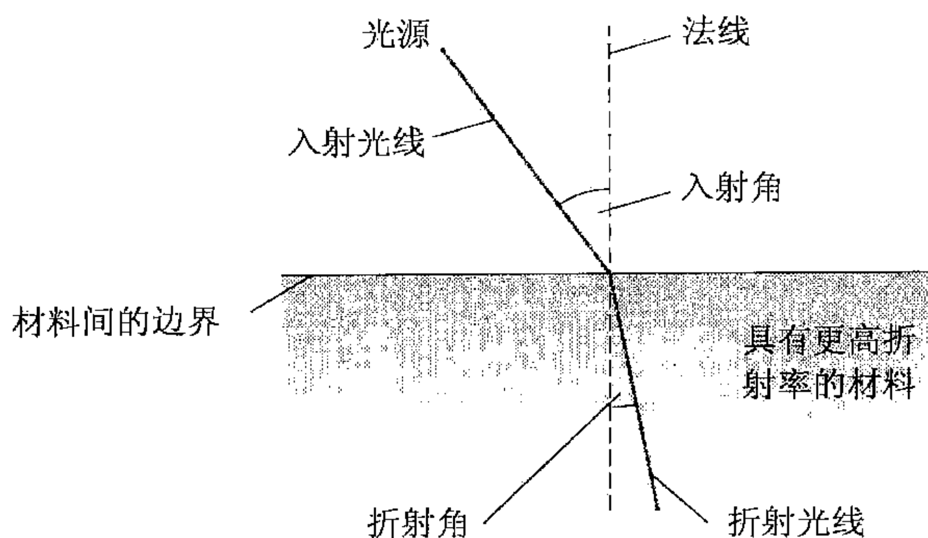


图10-9　折射光

为了准确地渲染透明物体，渲染软件必须知道相关材料的折射率，但这还不够，渲染软件还必须知道物体表面的哪一边表示物体的里面，而另一边为外面。光线是进入物体还是离开物体？获得这些信息的技术有时相当巧妙。例如，如果规定，从外部观察物体的时候，总是按逆时针顺序将多边形网格中多边形的顶点依次存放在一个列表中，那么通过给出的列表，我们可以很容易地得知面片的哪一边表示物体的外面。

10.4.2　裁剪、扫描转换和隐藏面的消除

现在着重考虑从场景图生成图像的过程。目前使用的方法正是在大多数交互式视频游戏系统中使用的技术。综合应用这些技术，形成了一个效果较好的方法，称为**渲染流水线**（rendering pipeline）。在本节的末尾我们将介绍这种方法的一些优缺点，在10.5节将探讨两种候选的方法。值得一提的是，渲染流水线处理不透明物体非常有效，因此折射就不是问题了。而且，它忽略了物体间的相互影响，因此现在我们就不必担心镜像和阴影问题。

渲染流水线首先确定包含相机能“看到”的物体（或部分物体）的三维场景中的区域。这个区域称为**视体**（view volume），它是锥体内的一个空间，这个锥体是由从投影中心出发向图像窗口边界延伸的直线所定义的（如图10-10所示）。

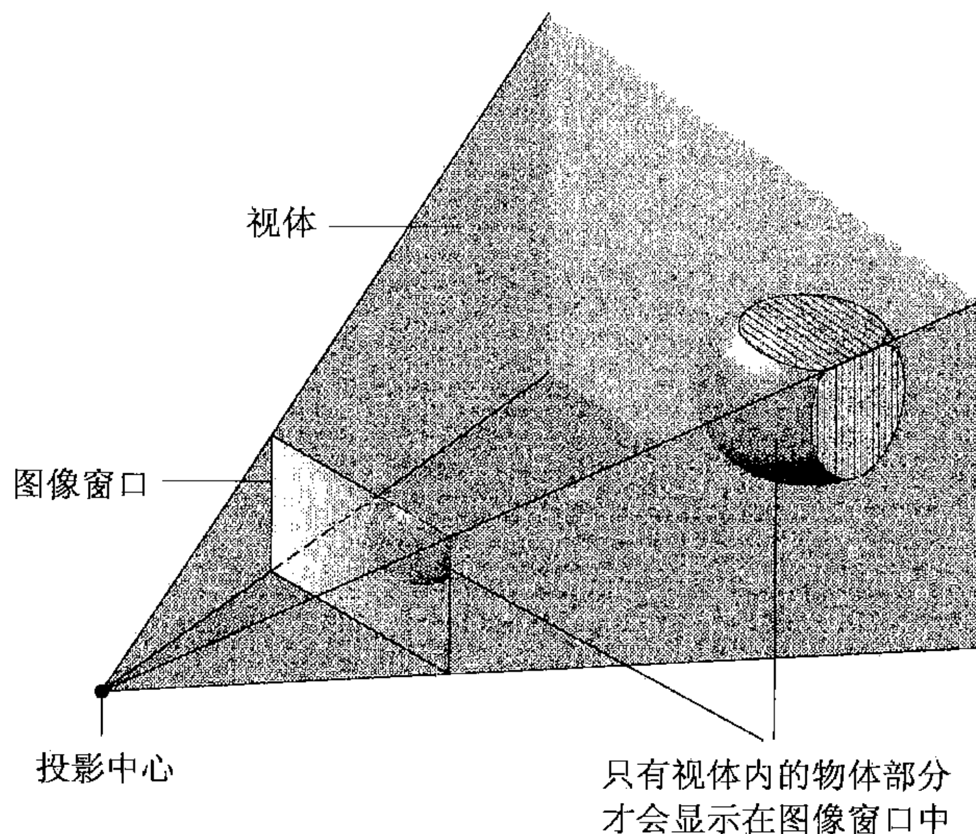


图10-10　确定在视体里面的场景区域

视体被确定下来以后，接下来就不用考虑那些与视体不相交的物体或物体部分了。毕竟，那部分的场景投影将落在图像窗口的外面，因此不会出现在最终的图像中。第一步就是去除完全在视体外面的物体。为了能够以流水线的形式进行处理，可以将场景图看作一个树型结构，其中处在场景不同区域的物体被存储在不同的分支上。因此，仅需忽略树中的整个分支，就可以去除大部分的场景图。

走　样

你是否注意到条纹衬衫和领带在电视屏幕上会出现奇怪的“闪光”？这是一个称为**走样**（aliasing）的现象产生的结果，当所期望的图像网格中的模式与组成图像的像素密度不匹配时，就会产生走样现象。例如，假设图像的一部分由黑白相间的条纹构成，但是所有像素的中心碰巧落在黑色条纹上，那么物体将被当作全黑色的来渲染。但是，如果物体稍微移动一下，所有像素的中心可能都落在白色条纹上，这样物体将会突然变成白色的。有多种方法可以改善这种令人心烦的效果。其中一种就是使用图像小区域的均值而不是精确的单个点来渲染每一个像素。

确定和去除那些与视体不相交的物体后，剩余的物体通过称为**裁剪**（clipping）的操作加以整理，它实际上就是去掉每个物体处在视体外面的部分。更准确点说，裁剪操作首先把每个平面片与视体边界进行比较，然后去除平面片落在外面的部分。最终得到完全都处在视体里面的多边形网格（可能是被裁剪的平面片）。

渲染流水线的下一步是确定剩余平面片上的点，这些点与最终图像中的像素位置相关。只有这些点将会对最终图像产生影响，认识到这一点是很重要的。如果物体上的细节落在像素位置的中间，那它就不能用1像素来表示，因此将不会显示在最终图像中。这就是像素数在数码相机市场被着重宣传的原因。像素点数越高，小的细节就越容易被拍摄在照片中。

把像素位置与场景中的点相关联的过程称为**扫描转换**（scan conversion）（因为它涉及把面片转化为水平的一行像素点，这称为扫描线）或**光栅化**（rasterization）（因为一组像素称为光栅）。扫描转换是由从投影中心出发穿过图像窗口中的每个像素的射线（放映机）来完成的，然后找到这些投影线与平面片的交点，最后，根据这些交点我们得到物体在图像上的外观。实际上，这些点在最终图像中是由像素表示的。

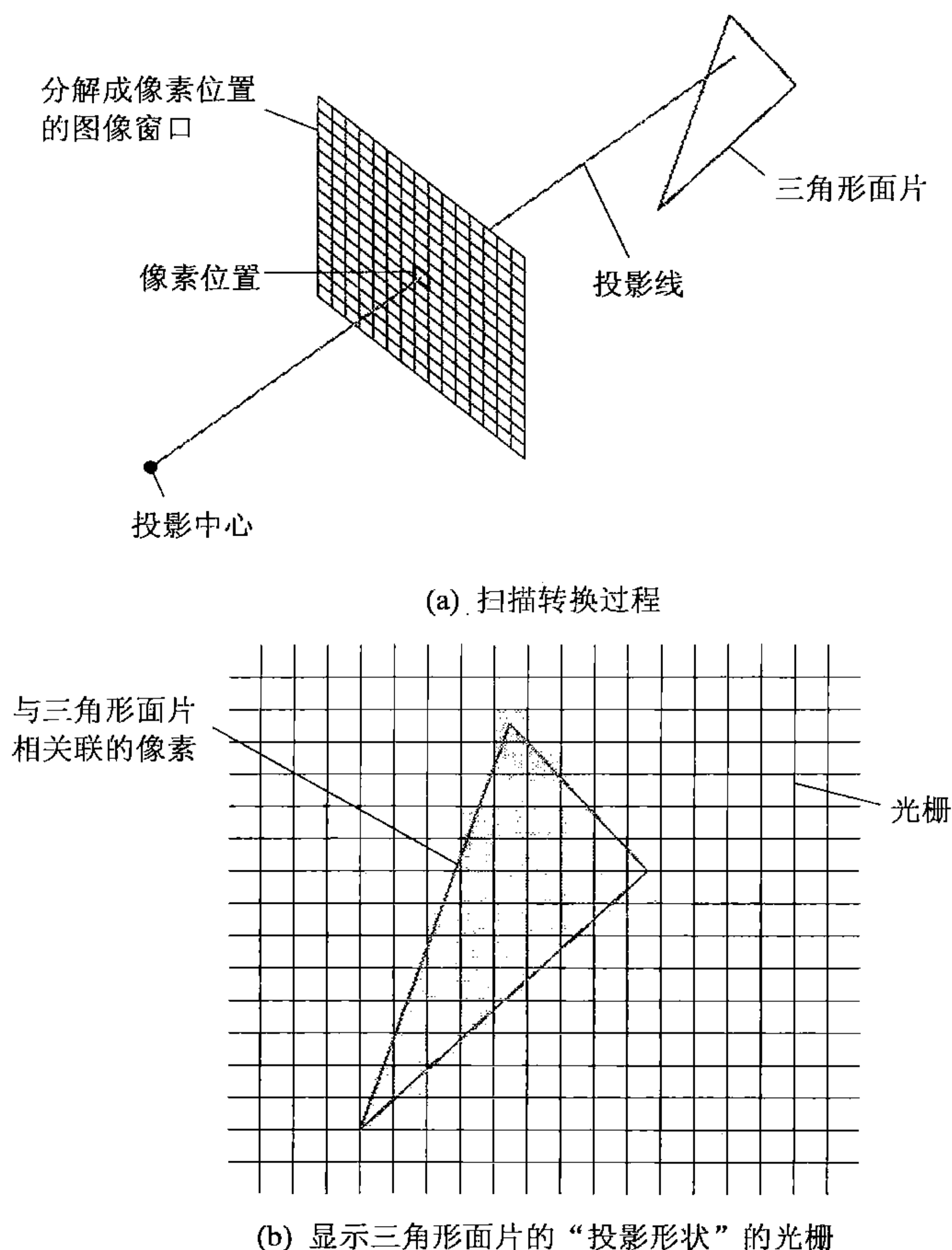
图10-11描述了单个三角形面的扫描转换。这幅图的a部分显示了如何通过投影线实现像素位置与面片上点的关联，b部分显示了通过扫描转换得到的面片的像素图像。像素的整个数组（光栅）由网格来表示，与三角形相关的像素已经被着色。注意，这个图还显示了当扫描转换的形状其特征点小于像素尺寸时，会产生变形。大多数拥有个人计算机的用户会经常在显示屏上看到这种锯齿状的边缘。

遗憾的是，整个场景（即使是单个物体）的扫描转换也不像扫描转换单个面片那样直观明了。这是因为，当涉及多个面片时，一个面片可能会遮盖住另一个面片。这样，即使投影线与平面片相交，面片上的这个点在最终图像上也不一定可见。识别和去除场景中被遮挡的点的处理称为**隐藏面消除**（hidden-surface removal）。

隐藏面消除的一个具体方法是使用**后面消除法**（back face elimination），也就是不考虑那些在多边形网格中表示物体“后面”的面片。注意后面消除是相对简单的，可以认为那些朝向背着相机的面片是处在物体后面的。

但是，后面消除法不能完全解决隐藏面消除问题。例如，想象一辆汽车在大楼前的场景，来自汽车和大楼的平面片将会投影到图像窗口的相同区域。在重叠发生的地方，最终存储在帧

缓冲区中的像素数据对应前景物体（汽车）的显示，而不是背景物体（大楼）。总之，如果投影线与多个平面片相交，那么最靠近图像窗口的面片上的点应该被渲染。



(a) 扫描转换过程

(b) 显示三角形面片的"投影形状"的光栅

图10-11　三角形面片的扫描转换

解决"前景/背景"问题的一个简单方法是众所周知的**画家算法**（painter's algorithm），就是根据相机到物体的距离，在场景中依次放置物体，然后先扫描转换最远的物体，允许较近物体的扫描转换结果覆盖先前的任何结果。遗憾的是，画家算法不能处理对象纠缠在一起的情况。比如，树的一部分可能在一个物体的后面，而另一部分可能在这个物体的前面。

对于"前景/背景"问题的更彻底的解决方案是集中考虑单个像素，而不是整个物体。一种常用的技术就是使用称为**z缓冲区**（z-buffer，也称深度缓冲区）的额外的存储区域，它包含了图像中每个像素的通道（也可以说是帧缓冲区中的像素通道）。z缓冲区中的每个位置被用来存储沿着相应的投影线从相机到物体间的距离，而这些物体用帧缓冲区中相应的通道来表示。只有当像素数据还没有放在帧缓冲区内，或者当前所观察物体的点比先前渲染的物体的点更近时（这是由帧缓冲区中所记录的距离信息决定的），借助于z缓冲区，并通过计算和存储像素的外观，"前景/背景"问题才得以解决。

更准确地说，当使用z缓冲区时，渲染过程可以按照如下步骤进行。为z缓冲区的所有通道设定一个值，表示从相机到要渲染物体间的最大距离。每当考虑渲染平面片上的任何一个新点时，首先将其与相机间的距离和z缓冲区中关联当前像素位置的值进行比较。如果距离小于z缓冲区中的值，则计算点的外观，在帧缓冲区中记录结果，用刚渲染点的距离替换z缓冲区中的相

应通道。（注意，如果此点的距离大于z缓冲区中相应的值，则不用作任何考虑，这也许是由于平面上的点太远了，或者它被已经渲染的更近的点遮住了。）

10.4.3 着色

扫描转换已经确定了要显示在最终图像中的平面片上的点之后，渲染任务就变成了决定这些点的外观方式的处理。这个过程称为**着色**（shading）。注意，着色涉及计算从相应的点投影到相机的光的特征，它取决于此点表面的朝向。毕竟，点表面的朝向决定了镜面反射光、散射光和环境光被相机拍摄的效果。

平面着色（flat shading）是着色问题的一个简单的解决方法，它是把平面片的方向作为其上每个点的方向，也就是说，假设每个面片上的表面都是平坦的。但是，最终图像将显示为有棱角的（如图10-12所示），而不像图10-8中显示得那样圆。从某种意义讲，平面着色生成的是多边形网格本身的图像，而不是用网格建模的物体图像。

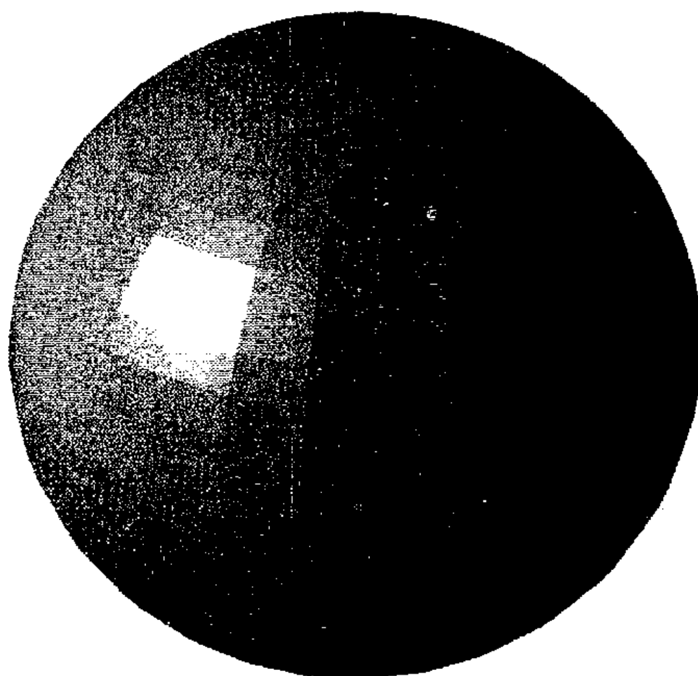


图10-12 当用平面着色渲染时，球的可能显示

为了生成更加逼真的图像，渲染过程必须将每个平面片的外观融合到平滑曲面显示的表面。这是通过估算每个渲染点最初表面的真实方向来完成的。

这个估计模式通常始于指示多边形网格顶点处表面朝向的数据。获得此数据有多种方法。一种方法就是在每个顶点处编码原始表面的方向，并把这个数据附着在多边形网格上，作为建模过程的一部分。这生成了一个带箭头的多边形网格，称为**法向量**（normal vector），这些箭头附着在每个顶点上。每个法向量沿垂直于原始表面的方向指向外部。这样的多边形网格可以想象成如图10-13所示的样子。（另一种方法是计算与顶点相邻的每个面片的朝向，然后使用这些朝向的"平均值"来估计顶点表面的朝向。）

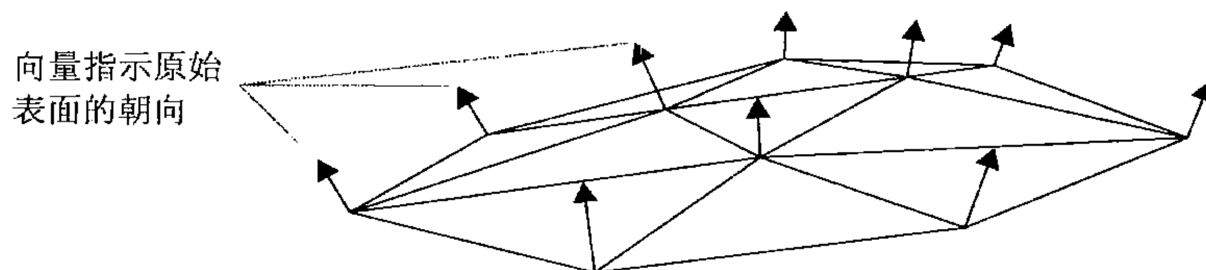


图10-13 在其顶点处带有法向量的多边形网格的概念视图

不管多边形网格顶点的原始表面朝向是如何获得的，这里有几种策略可以用来对基于这些数据的平面片进行着色，包括**Gouraud着色**和**Phong着色**，它们之间的区别是微妙的。二者首

先都使用面片顶点处的表面朝向信息近似估计沿着面片边界的表面朝向。然后Gouraud着色使用这些信息决定沿着面片边界的表面显示，最后，根据这个边界显示，插值估算出面片内部点处的表面的显示。相反，Phong着色则是根据沿着面片边界的表面朝向，插值估算出面片内部点处的表面朝向，然后只考虑显示问题。（总之，Gouraud着色将朝向信息转化为颜色信息，然后插值处理颜色信息；而Phong着色插值处理朝向信息，直至估计出点的朝向，然后将朝向信息转化为颜色信息。）结果是Phong着色更容易检测到面片内部的镜面反射光，因为它更易随表面朝向的变化而改变（参见10.4节结尾处的问题与练习3）。

最后，应该注意到可以扩充基础的着色技术，为表面添加纹理显示。例如，**凹凸映射**（bump mapping）的方法，实际上就是把生成的表面外观朝向加上一个小的变量，这样表面看起来就是粗糙的。更准确地说，凹凸映射在传统着色算法所使用的插值算法中加了一个自由度，这样整个表面看起来就是有纹理的，如图10-14所示。

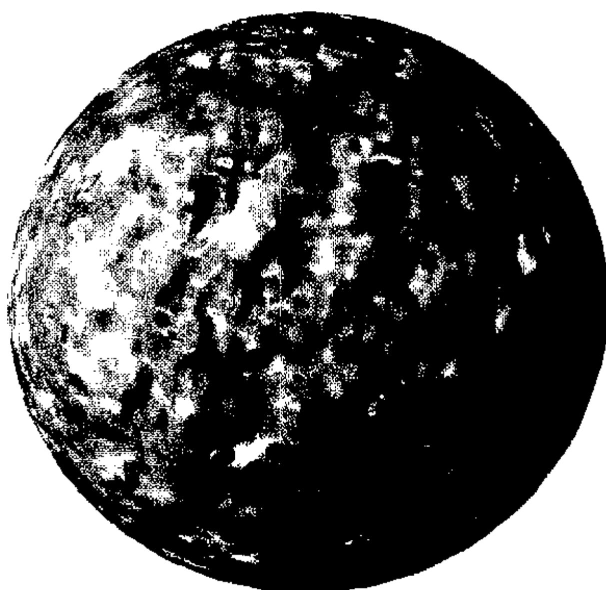


图10-14　使用凹凸映射渲染时，球的可能显示

10.4.4　渲染－流水线硬件

我们已经说过，渲染流水线就是依次执行裁剪、扫描转换、隐藏面的消除和着色处理这些操作。执行这些任务的高效算法是众所周知的，它们已经直接在电子电路中实现，通过超大规模集成电路（VLSI）芯片技术已经被微缩成芯片，自动完成整个渲染流水线。如今，即使是低廉的产品也具备每秒渲染几百万个平面片的能力。

大多数专门进行图形处理的计算机系统（包括视频游戏机）都在它们的设计中加入了这些设备。在更通用的计算机系统中，可以以**图形卡**（graphics card）或**图形适配器**（graphics adapter）的形式加入这种技术，它作为一个专门的控制器与计算机的总线连接（参见第2章）。这样的硬件大大减少了执行渲染处理所需的时间。

渲染－流水线硬件同样降低了图形应用软件的复杂度。从本质上讲，所有软件需要做的是向图形硬件提供场景图，然后硬件执行流水线步骤，把结果放置在帧缓冲区中。这样，从软件的角度来看，整个渲染流水线被缩减为作为抽象工具来使用硬件这一步骤。

我们再次以交互式视频游戏作为例子。为了初始化游戏，游戏软件把场景图传送给图形硬件，然后硬件渲染场景，把图像放置在帧缓冲区中，从这里图像就可以自动显示在监视屏上。当开始玩游戏时，游戏软件只需要更新图形硬件中的场景图，以反映出游戏场景的改变，而硬件则不断地渲染场景，每一次都把更新的图像放置在帧缓冲区中。

但是应该注意到，不同图形硬件的处理能力和通信特性是完全不同的。这就导致，如果像视频游戏这样的应用是在特定的图形平台上开发的，当移植到另一个环境中时，它就不得

不进行修改。为了减少对特定图形系统的依赖，标准的软件接口已经开发出来，它在图形硬件与应用软件之间起着重要的调节作用。这些接口包括把标准化命令转化为特殊图形硬件系统所需具体指令的软件例程。这些例子包含OpenGL（Open Graphics Library），它是由Silicon Graphics开发的非专有系统，广泛地用在视频游戏行业，Direct3D是微软为Windows环境开发的。

最后应该注意到，渲染流水线具有很多优点，但它也是有缺点的，其中最显著的就是流水线只实现了**局部照明模式**（local lighting model），这意味着流水线渲染的每个物体是独立于另一个物体的。也就是说，在局部照明模式下，每个物体是相对于光源被渲染的，仿佛它是场景中的唯一物体。结果是没有捕获到物体间的光交互（如阴影和反射）。与之相对，**全局照明模式**（global lighting model）就考虑了物体间的交互。在10.5节中我们将讨论两种实现全局照明模式的技术。现在我们仅需注意，这些技术超出了当前技术的实时能力。

但是，这并不意味着使用渲染流水线硬件的系统不能够产生一些全局照明的效果。实际上，巧妙的技术已经被开发出来，以克服局部照明模式所强加的一些限制。特别是可以在局部照明模式的环境里模拟**阴影**（drop shadow）的显示（投在地上的影子），通过建立投影物体的多边形网格的副本，把网格副本变成平坦的，然后放在地面上，涂以黑色。换言之，建模阴影，就好像它是另一个物体，可以用传统的渲染流水线硬件来渲染，生成阴影的幻觉。这种技术既在“大众水平”应用（如交互式视频游戏），也在“专业水平”应用（如飞行模拟）。

问题与练习

1. 总结镜面反射光、散射光和环境光之间的区别。
2. 定义术语裁剪和扫描转换。
3. Gouraud着色和Phong着色可以被总结为：
   Gouraud着色使用沿着面片边界的物体表面朝向决定沿着边界的表面的显示，然后把这些显示插值到面片的内部，生成特定点的显示；Phong着色通过对边界朝向的插值，计算面片内部点的朝向，然后使用这些朝向生成特定点的显示。物体的显示有何不同？
4. 渲染流水线的作用是什么？
5. 描述使用局部照明模式，如何模拟镜子中的反射。

*10.5 处理全局照明

研究者当前正在研究两种可以替代渲染流水线的方法，这两种方法都实现了全局照明模式，克服了传统流水线中局部照明模式的固有局限。其中一种方法是光线跟踪，另一种是辐射度。两种方法在处理时都极为精确但也很费时，在下文马上就可以看到。

10.5.1 光线跟踪

光线跟踪（ray tracing）本质上是沿着到达视点的光线进行反方向跟踪，以找到它的光源的过程。这个过程首先选择要渲染的像素，确定经过这个像素和投影中心的直线，然后跟踪沿着这条直线射入图像窗口的光线。这个跟踪过程沿着此直线进入场景，直至与物体相交。如果这个物体是光源，则终止光线跟踪过程，将像素渲染为光源上的一个点；否则，将计算物体表面的特性，以此得到入射光的方向，当前跟踪的光线是入射光被反射后产生的反射光线。然后继续跟踪入射光线向后找到光源，在这点上可能还会有另一条光线需要识别和跟踪。

图10-15给出了一个光线跟踪的例子，在图中可以看到被跟踪的光线向后穿过图像窗口，到达镜子的表面，从这里跟踪光线至一个有光泽的球，再向后到镜子，然后从镜子到光源。基于

从这个跟踪过程中获得的信息，图像中的像素应该显示为球上的一点，而这个球是被反射到镜子中的光源照亮的。

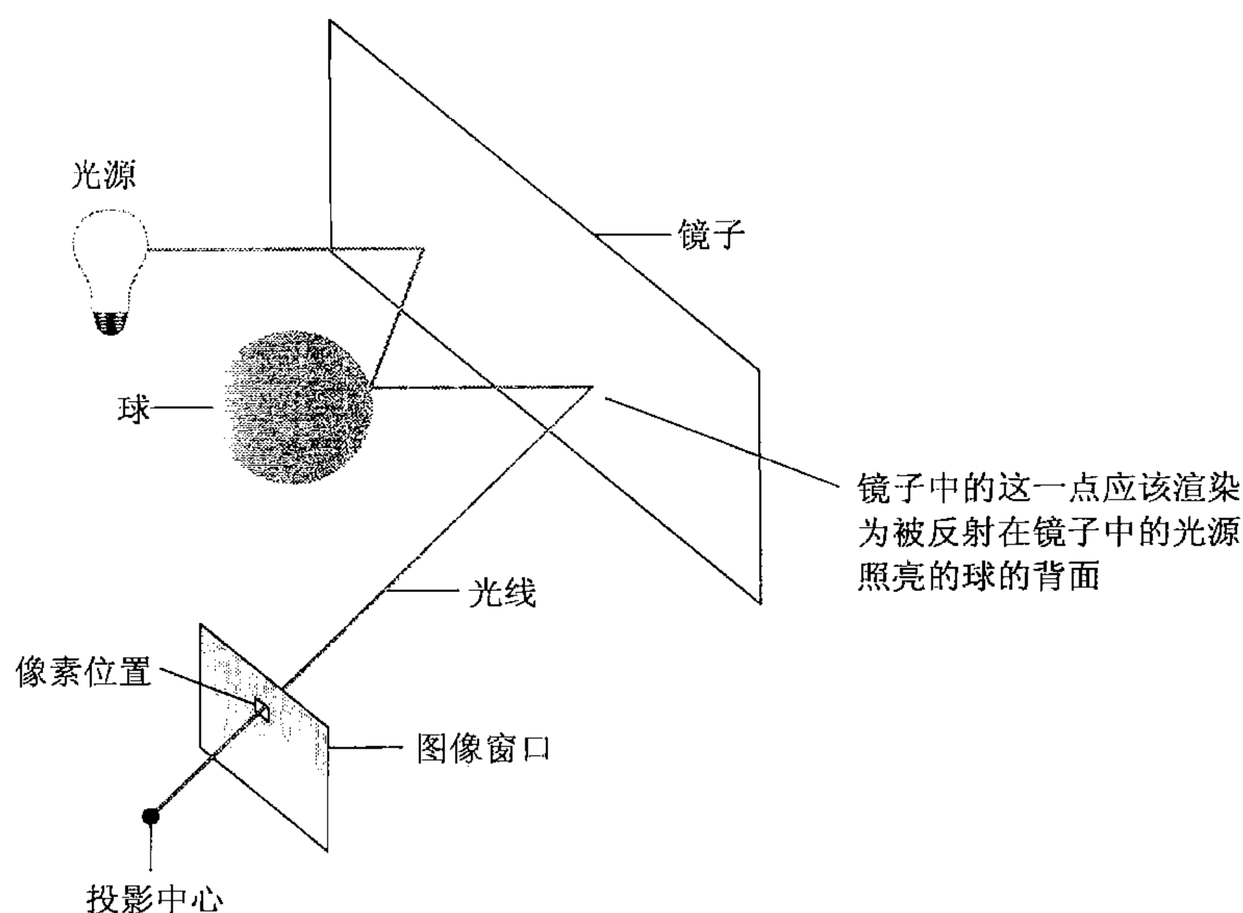


图10-15 光线跟踪

光线跟踪的一个缺点在于它仅跟踪镜面反射光。因此，通过这种方法渲染的所有物体往往都显示为有光泽的。可以使用**分布式光线跟踪**（distributed ray tracing）来去除这种效果。分布式光线跟踪与光线跟踪的区别在于，它并不仅仅跟踪从反射点向后的单条光线，而是同时跟踪从这点出发的多条光线，每条光线的延伸方向略有不同。

当涉及透明的物体时，基本光线跟踪的另一个变体是可以应用的。在这种情况下，每当向后跟踪光线至表面时，必须考虑两种效果，一种是反射，另一种是折射。例如，注意图10-1中巴斯光年（太空突击队员）头盔的透明显示，以及靠近头盔上表面的反射高光。在这种情况下，跟踪原始光的任务分成了两个：回溯追踪反射光和回溯追踪折射光。

光线跟踪通常是递归实现的，每次递归都跟踪光线到它的源头。第一次递归可能跟踪它的光线到一个有光泽的不透明的表面。在这点上此光线可能被确认为一条入射光线的反射光，计算这个入射光的方向，调用下一次递归去跟踪这个入射光线。第二次递归将执行类似的任务，去查找它的光线的源头。这个过程可能还会递归调用。

终止递归的光线跟踪有多种判断条件：被跟踪的光线与光源相交，被跟踪的光线可能存在于不与物体相交的场景中，或者是跟踪层数达到了预定的限制。还有另一种终止条件是基于表面的吸收特性的，如果表面是高吸收性的（如灰暗不光滑的表面），那么任何入射光线几乎都不会对表面的显示产生影响，光线跟踪停止，累积的吸收会产生相似的效果。也就是说，访问多个适度可吸收性的表面后，光线跟踪可以终止。

正是基于全局照明模式，光线跟踪才能避免传统渲染流水线中许多固有的限制。例如，隐藏面的消除和阴影检测通常可以在光线跟踪过程中自然地解决。遗憾的是，光线跟踪有一个很大的缺点，那就是费时。当跟踪每条反射光线至其源点时，需要的计算量迅速增大。（当涉及折射或者应用分布式光线跟踪时，这个问题是多方面的。）因此，光线跟踪没有在“大众水平”的实时系统中实现（如交互式视频游戏），相反可以在“专业水平”的应用中找到，这些应用对实

时性要求不高（如电影工作室中使用的图像软件）。

10.5.2 辐射度

另一种可以取代传统渲染流水线的方法就是**辐射度**（radiosity）。光线跟踪通过跟踪单条光线，采用点到点的方法；而辐射度通过考虑平面片对之间辐射光能的总和，采取了更加区域化的方法。这个辐射的光能本质上就是散射光。从一个物体辐射的光能或者是物体产生的（如光源这种情况），或者是物体反射的。然后通过考虑从其他物体接收的光能，来决定每个物体的显示。

从一个物体辐射的光对另一个物体显示的影响程度是由**形状因子**（form factor）参数来衡量的。在场景中要渲染的每对面片都关联唯一的形状因子。这些形状因子考虑了面片间的几何关系，比如分开的距离和相对朝向等因素。为了显示场景中的一个面片，需要计算此面片所接收的来自场景中其他面片的光能总量，每次计算都要使用一个合适的形状因子，将结果结合，产生每个面片的单个颜色和强度。然后使用类似于Gouraud着色的技术，将这些值插值到相邻的面片中间，获得光滑没有棱角的表面。

因为要考虑许多面片，所以辐射度的计算量是很大的。因此，与光线跟踪一样，辐射度的应用无法满足当前消费市场上图形系统的实时要求。辐射度的另外一个问题在于，由于它处理的单元包含整个面片，而不是单个的点，所以它捕获不到镜面反射光，这就导致用辐射度渲染的所有表面往往是灰暗的。

但是，辐射度的确有它的优点。首先，使用辐射度决定物体的显示是不受相机影响的。这样，计算了一个场景的辐射度以后，当相机位置改变时，场景的渲染将会快速完成。其次，辐射度捕获了光的许多细微特征，如**渗色**（color bleeding），即一个物体的颜色影响周围其他物体的色调。因此，辐射度有它的特定应用。其中之一就是用在建筑设计的图形软件中。实际上，建筑内的光主要是由散射光和环境光组成，镜面反射光的影响不大，并且新的相机位置可以得到有效处理，这一事实意味着建筑师能很快从不同的视角看到不同的房间。

问题与练习

1. 为什么光线跟踪是沿着光线向后（从图像窗口到光源），而不是向前（从光源到图像窗口）？
2. 直接的光线跟踪和分布式光线跟踪间的区别是什么？
3. 辐射度的两个缺点是什么？
4. 光线跟踪和辐射度在哪些方面是相似的？在哪些方面是不同的？

10.6 动画

我们现在转向计算机动画这个课题，它是使用计算机技术来生成和显示呈现运动的图像。

10.6.1 动画基础

我们先介绍一些基本的动画概念。

1. 帧

动画是通过快速连续地显示图像序列——称为**帧**（frame）——获得的。这些帧捕获了相同时间间隔内变化场景的显示。这样，它们的顺序展示就产生了一直观看连续场景的错觉。在制片厂显示的标准速率是每秒24帧，广播视频的标准是每秒60帧（由于每隔一帧的视频帧被用来与前一帧混合，以产生细节完整的图像，所以视频也可以被归类为每秒30帧的系统）。

帧可以由传统的照片生成，也可以利用计算机图形学生成。而且，这两种技术还可以结合应用。例如，2D图形软件经常被用来修改通过照片获得的图像，消去支撑金属丝、添加图像、和创建**扭曲**（morphing）效果（这是一个物体看起来要转化成另一个物体的处理过程）。

让我们一起来深入研究扭曲，它使动画变得更有趣。构建扭曲效果首先要确定把扭曲序列括在一起的一对关键帧。一个是扭曲处理之前的最后一帧的图像；另一个是扭曲处理之后的第一帧图像。（在传统的电影制作中，这需要“拍摄”两个动作序列：一个是在扭曲之前发生；另一个在扭曲后进行。）把扭曲前与扭曲后帧中的相似特征相关联的点称为**控制点**（control point）。扭曲处理是指应用数学技术，以控制点作为基准，逐渐把一幅图像转变为另一幅图像的处理过程。通过记录扭曲过程中所产生的图像，得到了一个简短的人工合成的图像序列，把它填充到当初定义的那时关键帧之间，就创建了扭曲效果。

Kineographs①

Kineographs是以书页作为帧的动画，它通过书页快速翻动来模拟帧的播放。通过这部分内容，你能制作你自己的Kineographs（假定你还没有在书的空白处填满涂鸦）。在第一页的空白处放置一个圆点，然后在第三页放置另一个圆点，位置与第一页的稍有不同。在每个连续的奇数页上重复这个过程，直到你到达了书的末尾处。现在，快速翻动书页，观察点的跳动情况。一转眼，你已经制作了自己的Kineographs，也许这是你迈向动画事业的第一步。作为运动学的一个实验，你可以试着画粗线条人物替代简单的圆点，使得粗线条人物看起来像在走路。现在，动力学实验是通过产生水滴撞击地面的图像来进行的。

2. 故事板

一个典型的动画项目始于**故事板**（storyboard）的创建，故事板是一个二维图像序列，用关键点处显示的场景草图的形式讲述一个完整的故事。故事板的最终角色取决于动画项目是用2D技术，还是用3D技术实现的。在使用2D图形的项目中，故事板一般会转变成帧中的最终场景，就像20世纪20年代迪士尼工作室所做的一样。在那个时候，艺术家（称动画大师）将把故事板细化为详细的帧——**关键帧**（key frame），它决定了动画固定时间间隔的角色和场景的显示。助理动画师将绘制出填充关键帧之间间隔的另外的帧，以使动画看起来连续而平滑。这个填充间隔的处理称为**中间存在**（in-betweening）。

与以前不同的是，现在的动画制作人员使用图像处理和2D图形软件绘制出关键帧，大量的中间存在的处理是自动的，所以已经不存在助理动画师这个角色。

3. 3D动画

大多数视频游戏动画和功能完备的动画产品现在是使用3D图形创建的。在这些情况下，项目仍旧先创建由二维图像组成的故事板。但是，与发展成为2D图像项目的最终产品不同，故事板被用作构建三维虚拟世界的蓝图。当其中的物体按照脚本移动或在视频游戏中行进时，这个虚拟世界将不断地被“拍摄”。

也许我们应该停下来弄清楚物体在计算机生成的场景内移动的含义。记住，“物体”实际上是存储在场景图中的数据集合。这个集合中的数据是一些指示物体位置和朝向的值。这样，“移动”一个物体仅需通过改变这些值就可以完成。已经作出这些改变以后，新的值将被用在渲染处理中。从而，物体将在最终的二维图像中显示为已经移动。

① 指以书做动画的一种方法。——译者注

模 糊

在传统的摄影领域，人们做出了大量的努力，以生成快速运动物体的清晰图像。在动画领域，相反的问题产生了。如果描述运动物体的序列帧中的每一帧都把物体渲染成清晰的图像，那么运动可能显示为急动的。然而，清晰的图像是创建帧时的自然副产品，因为场景图中的每幅图像都是静止的。这样，动画制作人员经常需要手动扭曲计算机生成帧中的运动物体的图像。有一种技术称为**超取样**（supersampling），它产生多幅图像，其中运动物体只是稍微地移动，然后重叠这些图像生成单帧。另一种技术是改变运动物体的形状，使它看起来像是沿着运动方向延伸的。

10.6.2 运动学和动力学

3D图形场景中的运动究竟是自动的，还是受动画制作者控制的，这个自动化的程度随应用的不同而变得不尽相同。当然，目标是整个过程的自动化。为了实现这个目标，许多研究已经直接面向找寻能识别和模拟自然发生现象的运动的方法。机械学中的两个领域被证明在这一点上特别有用。

一个是**动力学**（dynamics），它通过应用物理定理计算力作用于物体的效果，来描述物体的运动。例如，除了位置外，场景中的物体可能被赋予运动的方向、速度和质量。然后使用这些数据来决定重力或与其他物体的碰撞对物体产生的影响，为软件计算下一帧中物体的合理位置提供依据。

考虑创建一个描述容器中水的晃动的动画序列。我们可以使用场景图中的粒子系统来表示水，每个粒子表示一个小单位的水。（想象水由石弹珠大小的大“分子”构成。）那么，当容器从一边摇到另一边时，我们可以应用物理定理计算粒子重力的影响，以及粒子间的相互作用。这样可以计算出固定时间间隔内每个粒子的位置，通过使用外层粒子的位置作为多边形网格的顶点，我们能够获得描述水表面的网格。在模拟过程中，通过不断地“拍摄”这些网格，就得到了动画。

用来模拟运动的另一个机械学分支是**运动学**（kinematics），它根据物体各部分间的相对运动方式，来描述物体的运动。当模拟有关节的角色时，运动学应用的优势就特别显著，因为这需要移动像手臂和腿之类的附属物。与计算肌肉和重力施加的力的影响相比，模拟关节运动模式更容易对运动进行建模。因此，当确定弹球的路径时，动力学可能是可选的技术；而模拟人物手臂的运动将使用运动学计算出合理的肩膀、肘和手腕的旋转度。因此，许多模拟生物特性的研究集中在解剖学的问题上——关节与附属物的结构是如何影响运动的。

应用运动学的一个典型范例是：首先用粗线条人物图来表示人物，这个粗线条人物图模拟了被描述的人物骨骼结构；然后用多边形网格覆盖人物的每个部分，表示围绕这个部分的人物表面，并且建立相应的规则，决定相邻的网格相互连接的方式。通过重新定位关节在骨骼结构中的位置，就能够操纵人物（通过软件或动画制作人来操纵），就像一个人操纵提线木偶时使用的方式。“线”连接到模型的点称为**关节变量**（avars），它是“articulation variables”的缩写（或“animation variables”的缩写）。

在实际应用中，关节变量不仅仅用来控制骨骼关节的位置。例如，《玩具总动员》中名为Woody的牛仔玩偶（如图10-1所示）仅在脸部就有将近100个关节变量，动画制作者可以通过改变相应变量的值来表达牛仔玩偶的情感，或根据所说的话语变动口形。

许多运动学应用的研究已经朝着开发算法的方向发展，以自动计算模拟自然发生运动的附属物位置的序列。沿着这些方向，生成逼真的行走序列的算法现在已经产生了。

但是，基于运动学的大量动画还是通过指示人物穿过关节－附着物位置的预设序列而产生的。这些位置可能是由动画制作者创造性地给出，或是由**运动捕捉**（motion capture）而获得，当生物模型执行相应的动作时，这个运动捕捉记录了模型的位置。更准确地说，在人身体上的主要关节应用反光带后，人投掷棒球的动作就可以从多个角度拍摄。然后，当人进行投掷动作时，通过观察各种照片中反射带的位置，人的手臂和腿的精确位置就能够被标识出来。然后将这些位置传送给动画中的人物。

10.6.3 动画制作过程

动画研究的最终目标是自动化整个动画过程。想象一下软件（给定合理的参数）能自动生成所需的动画序列。当今，影视公司通过单个虚拟“机器人”生成了人群图像、战争场景和惊逃的动物，这些“机器人”在场景中自动移动，每个都执行分配给它的任务，这充分印证了人类在动画制作领域中取得的进步。

当拍摄和《指环王》三部曲中的《魔鬼幻想部队》人类时，有趣的情况发生了。每一个屏幕上的战士都被建模成不同的“智能”物体，它们具有自己的体型特征和随机赋予的个性，这种个性赋予它一个攻击或是逃跑的倾向。在电影*Helms Deep*（第二部）的战争模拟测试中，魔鬼的逃跑倾向设置得过高，当它们刚遇到人类战士时，就逃跑了。（这也许是虚拟的“临时演员”认为工作太危险的第一例。）

当然，如今许多动画还是由动画软件来制作的。但是，不同于20世纪20年代的手工绘制二维帧，如今这些动画制作人员使用软件操纵场景图中的三维虚拟物体，这种方式让人想起前面讨论运动学时介绍的控制提线木偶的方式。用这种方式，一位动画制作者能够创建一系列的虚拟场景，这些场景然后被“拍摄”成动画。在某些情况下，使用这种技术来产生关键帧的场景，然后当软件应用运动学和动力学把场景图中物体从一个关键帧场景位置移到下一个关键帧场景位置时，再使用软件通过自动渲染场景来生产中间存在帧。

随着计算机图形学研究的进步以及技术的持续发展，肯定会有更多的动画制作过程变成自动化的。是否有一天动画制作者这一角色将不复存在，人类演员和物理布景也将成为过去，针对这一点尚无定论，但许多人都认为这一天不再遥远。实际上，与把无声电影转化为有声电影相比，3D图形学对影视产业的影响可能更为深远。

问题与练习

1. 人类看见的图像往往在人类的感知上约有200 ms的视觉暂留，基于这个近似值，每秒必须呈现多少幅图像，才能产生动画？这个近似值是如何对应于电影中使用的每秒的帧数的？
2. 什么是故事板？
3. 什么是中间存在？
4. 定义术语运动学和动力学。

复习题

1. 下列哪个是2D图形学的应用，哪个是3D图形学的应用？
   a. 设计杂志页面的布局。
   b. 用微软绘图工具绘制图像。
   c. 为视频游戏从虚拟世界中生成图像。
2. 3D图形学中的哪些术语对应于下列传统摄影领域的术语？请解释你的答案。
   a. 胶卷。
   b. 取景器中的矩形。
   c. 被拍摄的场景。

3. 当使用透视投影时，场景中的球在什么条件下不会在投影平面上产生一个圆圈？
4. 当使用透视投影时，直线段的图像能变成投影平面上的曲线段吗？证明你的答案。
5. 假设8英尺[①]直柱的一端距离投影中心4英尺，而且假设从投影中心到直柱一端的直线与投影平面相交于一点，这一点距离投影中心1英尺，如果柱体与投影平面平行，那么在投影平面里柱子的图像长度是多少？
6. 说明平行投影和透视投影之间的区别。
7. 说明图像窗口和帧缓冲区之间的关系。
8. 应用3D图形学产生电影和应用3D图形学产生交互式视频游戏的动画，二者之间有什么明显的不同？
9. 说出物体的一些特性，这些特性可能出现在3D图形场景使用的这个物体的模型中，说出可能不会出现在模型中的一些特性，并解释你的答案。
10. 说出一些未被只含多边形网格模型捕获的物体的物理特性。（这样，单独的多边形网格没有构建一个完整的物体模型。）如何把这些物理特性中的一个添加到模型中？
11. 三维空间中的任意4个点能是多边形网格中的面片的顶点吗？解释你的答案。
12. 下面的每个集合都表示了一个多边形网格中面片的顶点（使用传统的直角坐标系），描述网格的形状。

    面片1：(0,0,0)　(0,2,0)　(2,2,0)　(2,0,0)
    面片2：(0,0,0)　(1,1,1)　(2,0,0)
    面片3：(2,0,0)　(1,1,1)　(2,2,0)
    面片4：(2,2,0)　(1,1,1)　(0,2,0)
    面片5：(0,2,0)　(1,1,1)　(0,0,0)
13. 下面的每个集合都表示了一个多边形网格中面片的顶点（使用传统的直角坐标系），描述网格的形状。

    面片1：(0,0,0)　(0,4,0)　(2,4,0)　(2,0,0)
    面片2：(0,0,0)　(0,4,0)　(1,4,1)　(1,0,1)
    面片3：(2,0,0)　(1,0,1)　(1,4,1)　(2,4,0)
    面片4：(0,0,0)　(1,0,1)　(2,0,0)
    面片5：(2,4,0)　(1,4,1)　(0,4,0)
14. 设计表示长方体的多边形网格，使用传统的直角坐标系来表示顶点，绘制出一幅草图表示你的解决方案。
15. 使用不超过8个三角片，设计多边形网格，去近似表示半径为1的球的形状。（只有8个面片，你的网格将是非常粗糙的近似球形，但目的是帮助你理解什么是多边形网格，而不是产生球的精确表示。）使用传统的直角坐标系表示面片的顶点，绘制出网格的草图。
16. 为什么下面4个点不是平面片的顶点？

    (0,0,0)　(1,0,0)
    (0,1,0)　(0,0,1)
17. 假设顶点（1,0,0）、（1,1,1）和（1,0,2）是平面片的顶点，下面哪个线段是面片表面的法线？
    a. 从（1,0,0） 到（1,1,0）的线段
    b. 从（1,1,1） 到（2,1,1）的线段
    c. 从（1,0,2） 到（0,0,2）的线段
    d. 从（1,0,0） 到（1,1,1）的线段
18. 说出程序化模型的两种“类型”。
19. 在建模过程和渲染过程之间，哪一个更是
    a. 标准化任务？
    b. 以计算为主的任务？
    c. 创造性任务？

    证明你的答案。
20. 下列哪一个可能在场景图中表示？
    a. 光源
    b. 不动的道具
    c. 人物/演员
    d. 相机
21. 在何种意义上说场景图的创建是3D图形学处理的关键步骤？
22. 场景图中的相机可能改变位置和朝向，这个问题引入了何种复杂性？
23. 假设带有顶点（0,0,0）、（0,2,0）、（2,2,0）和（2,0,0）的平面片是平滑且有光泽的。如果一条从点（0,0,1）发出的光线，在（1,1,0）处入射表面，反射光将经过下列哪个点？
    a.（0,0,1）
    b.（1,1,1）
    c.（2,2,1）
    d.（3,3,1）

① 1英尺=0.3048米。——编者注

24. 假设一个浮标上支撑着一个离静止水面高10英尺的灯，如果有一观察者离浮标15英尺，离水面高5英尺，在水面的哪个点上，观察者将看到灯的反射？
25. 如果一条鱼在静止水面下游动，观察者从水面上看鱼，从观察者的位置来说，鱼将显示为：
    a. 在它的真实位置的上方且朝向背景方向；
    b. 在真实的位置；
    c. 在它的真实位置的下方且朝向前景方向。
26. 假设点（1,0,0）、（1,1,1）和（1,0,2）是平面片的顶点，并且从物体外面观看时顶点是以逆时针顺序依次排列。在每种情况下（从物体外或物体内部观看），指明从所给出的点发出的光线是否会与面片的表面相交。
    a.（0,0,0）
    b.（2,0,0）
    c.（2,1,1）
    d.（3,2,1）
27. 举一个例子，说明视体外的物体能够出现在最终图像中，并解释你的答案。
28. 描述z缓冲区的内容和用途。
29. 在针对隐藏面消除的讨论中，借助z缓冲区我们描述了解决“前景/背景”问题的过程。用第5章介绍的伪代码表示这个过程。
30. 假设物体的表面被交替的橙色和蓝色竖条纹覆盖，每一个条纹的宽度都是1厘米，如果这个目标被放置在场景中，这样像素的位置与物体上的占据2厘米空间的点相关联，在最终图像中，物体的可能显示是什么？解释你的答案。
31. 虽然纹理映射和弹性映射都是与表面“纹理”相关的方法，但它们是相当不同的技术，用一小段话来对它们进行比较。
32. 列出渲染流水线的4个步骤，并给出简要说明。
33. 使用硬件/固件实现渲染流水线有哪些优点？
34. 设计为交互式视频游戏的计算机的硬件与通用PC的硬件在哪些方面不同？
35. 传统渲染流水线的主要限制是什么？
36. 局部照明模式与全局照明模式之间的区别是什么？
37. 光线跟踪与传统的渲染流水线相比有哪些优点，有哪些缺点？
38. 分布式光线跟踪与传统的光线跟踪相比有哪些优点，有哪些缺点？
39. 辐射度与传统的渲染流水线相比有哪些优点，有哪些缺点？
40. 如果用传统光线跟踪生成的场景图像与用辐射度生成的相同场景的类似图像相比较，两幅图像有何异同？
41. 电影院放映的90分钟动画产品需要多少帧？
42. 描述粒子系统是如何被用来产生火苗闪烁的动画的。
43. 当创建一个描述单个物体在场景中移动的动画序列时，解释z缓冲区的使用有何帮助。
44. 当今的动画制作者与以前的动画制作者的工作有哪些不同？

社会问题

下列问题有助于你分析一些与计算领域相关的道德、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 假设计算机动画到达了在影视行业中不再需要真实演员的地步，那结果将是什么？不再有“电影明星”会带来什么样的影响？
2. 随着数码相机和相关软件的发展，大众已经可以改变和制作照片。这将给社会带来何种变化？会引起哪些道德和法律问题？
3. 照片所有权的程度是什么？假设一个人在网站上放置了他的照片，另一些人下载了这张照片并修改了它，因此，主体处于妥协的状况，后来流行的是改变后的版本，照片的主体应该拥有什么追索权？
4. 帮助开发暴力视频游戏的程序员应该对此游戏产生的任何后果负有何种程度的责任？是否应该限制儿童接触此类游戏？如果是的话，应如何限制以及由谁来限制？对社会上其他一些特殊的团体（如罪犯），应该采取何种限制？

课外阅读

Angel, E. *Interactive Computer Graphics, A Top-Down Approach Using OpenGL*, 5th ed. Boston, MA: Addison-Wesley, 2009.

Bowman, D. A., E. Kruijff, J. J. LaViola, Jr., and I. Poupyrev. *3D User Interfaces Theory and Practice*. Boston, MA: Addison-Wesley, 2005.

Hill, Jr., F. L. and S. Kelley. *Computer Graphics Using OpenGL*. 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2007.

McConnell, J. J. *Computer Graphics, Theory into Practice*. Sudbury, MA: Jones and Bartlett, 2006.

Parent, R. *Computer Animation, Algorithms and Techniques*, 2nd.ed San Francisco, CA: Morgan Kaufmann, 2008.

第11章 人工智能

本章探讨计算机科学的一个分支：人工智能。尽管该领域仍然处于发展的初期阶段，但它已经产生了一些令人惊讶的结果，例如，机器人象棋大师，用来学习和推理的计算机，协调一致来完成一个共同的目标（如赢得一场足球比赛）的多个机器。在人工智能领域，今天的科学想象也许在明天就会变成现实。

本章内容

人工智能是计算机科学的一个领域，旨在设法建造自主的机器——无需人为干预就能完成复杂任务的机器。这个目标要求计算机能够感知和推理，虽然这两种能力属于常识行为，对于人脑来说是与生俱来的，但对于机器来说却是有困难的。结果是该领域的工作一直富于挑战性。本章就来探讨这个广阔研究领域中的一些主题。

11.1 智能与机器

人工智能领域十分广阔，并且与其他学科相融合，如心理学、神经学、数学、语言学以及电子与机械工程等学科。为了集中思绪，我们先来考虑智能体的概念以及智能体可能呈现的智能行为类型。实际上，人工智能的许多研究都可按智能体的行为来分类。

11.1.1 智能体

智能体（agent）是对环境的刺激做出响应的一种“装置”。很自然地人们会把一个智能体想象为一个像机器人这样的单个机器，尽管它可以有别的形式，如一架自动飞机、交互式视频游戏里的一个角色，或是通过因特网通信的进程（可能作为客户机、服务器或对等机）。大多数智能体具有传感器和效应器，前者接收来自环境的数据，后者对环境做出反应。常见的传感器包括麦克风、摄像机、距离传感器以及空气或土壤采样设备等，效应器的例子有车轮、腿、翅膀、夹子以及语音合成器。

很多人工智能的研究可以在构建智能体的环境中进行描述，这意味着智能体效应器的动作必须对通过传感器接收的数据做出合理的响应。按照响应的级别，我们可以对这些研究进行分类。

最简单的响应是映射行为，这只是对输入数据的一个预定的响应。更高级的响应用于获取更智能的行为。例如，我们可以赋予智能体以环境知识，要求其相应地调整行为。投掷棒球的过程在很大程度上是映射行为，但决定怎样扔，向哪个方向扔，就需要对当前环境有一定的了解（如此时一个人出局，跑手在一垒和三垒）。怎么样存储、更新和获取这种现实世界的知识，然后最终应用到决策过程中，一直是人工智能领域具有挑战性的问题。

如果我们希望智能体的目标是赢得一场棋赛或是通过一条拥挤的通道，那么就需要另一种层次的响应。这种有目标性的行为需要智能体的响应（或是一系列的响应）应当是周密考虑的结果，构成一个行为计划，或是在当前的各种选项中选取最好的行为。

在有些情况，随着智能体不断学习，它的响应能够不断地得到改进。这可以采取不断发展**过程性知识**（procedural knowledge）的形式（学习"怎样"），或者储备**陈述性知识**（declarative knowledge）的形式（学习"什么"）。学习过程性知识涉及一个反复试验的过程，在这个过程中，智能体从出错受罚、正确受奖的过程中学习适当的反应。根据这个方法开发了一些智能体，它们能够随着时间的推移逐步提高在跳棋和国际象棋等竞赛性游戏中的能力。学习陈述性知识通常采取的形式是扩充或变更智能体的知识库里的"事实"。例如，一个棒球运动员必须不断地重复调整其知识数据库（虽然还是一人出局，但现在跑垒手在第一垒和第二垒），从中对将来的事件做出合理响应。

一个智能体要对刺激做出合理的响应，就必须"理解"由其传感器接收的刺激。也就是说，智能体必须能够从其传感器产生的数据里提取信息，即智能体必须能够感知。有些情况下，这是一个简单的过程。从一个陀螺仪获取的信号很容易就能编码成适合计算的形式，以确定响应。但是有些情况下，从输入数据提取信息并不容易，例如理解语音和理解图像就很难。同样，智能体也必须能够以与效应器兼容的方式表达它们的响应。这可以是一个简单的过程，也可能要求智能体把其响应表达为一个完整的口语句子——这意味着智能体必须生成语音。所以，像图像处理和分析、自然语言理解以及语音的生成这些主题都是重要的研究领域。

我们这里标识的智能体的属性既表示以前的研究范畴，又表示当今的研究范畴。当然，它们彼此之间并不是完全无关的。我们希望最终能够开发出处理所有这些属性的智能体，产生出能够理解来自环境的数据，并通过学习过程开发新的响应模式的智能体，学习的目的是最大限度地提高智能体的能力。然而，通过孤立各类推理行为并独立研究它们，研究人员获得了一个立足点，由此入手，可以与其他领域的发展相结合，产生更加智能的智能体。

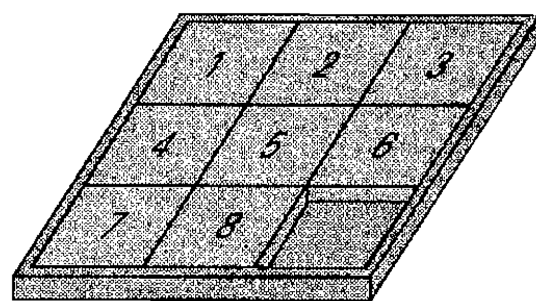
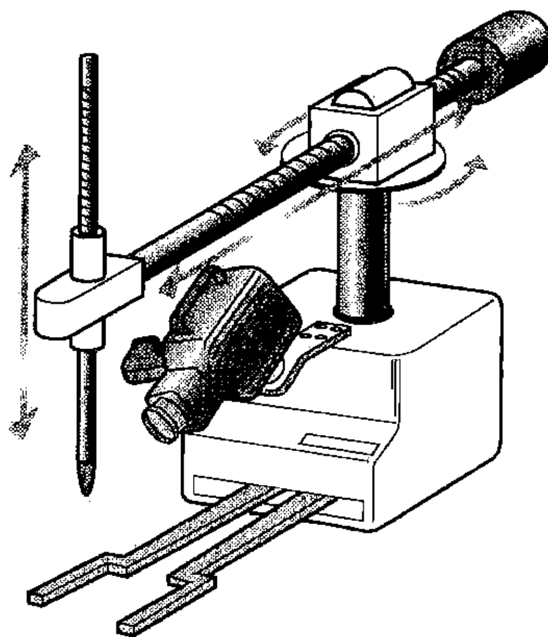


图11-1 8块拼图的最终布局

本节的最后我们介绍一个智能体，也为11.2节和11.3节的讨论提供一个背景。该智能体是为解决八数码游戏（eight-puzzle）而设计的，该游戏由8个小方块组成，标号为1～8，放置在一个3行3列总共可容纳9个小方块的框架内（如图11-1所示）。这样，框内的方块间有个空位，挨着空位的方块可以移动，允许框内的方块随意排布。问题是要把杂乱排布的方块移回到它们的初始位置（如图11-1所示）。


图11-2 解决八数码游戏的机器

我们的智能体采用如图11-2所示的这样一种装置，该装置配备有一个夹具、一个摄像机和一个带橡皮头的指杆，橡皮是为了推移东西时不会打滑。当首次开启该智能体时，夹具会一张一合，好像索要这个拼图一样。当我们把一个随意

排布的拼图放进夹具时，夹具就会把它夹住。不一会儿，机器的指杆降低，并开始在框架内推移方块，直到所有的方块恢复到原始位置。这时，机器就放开拼板并自己关掉电源。

这个解决八数码游戏的机器展现了已经提到过的两个智能体的属性。第一，它必须能够感知，就是必须从摄像机拍摄的图像中获取当前拼图的状态。我们将在11.2节阐述理解图像的问题。第二，它必须开发和实现一个达到目标的计划。这些问题将在11.3节中阐述。

11.1.2　研究方法

要对人工智能领域作出客观的评价，应该知道对该领域的研究存在两种路线。一种可以称为工程线路，即研究人员侧重于开发展示智能行为的系统。另一种可以称为理论路线，即研究人员会尝试从计算的角度来研究动物（尤其是人类）的智能。我们通过考虑这两种路线的执行方式，来说明这种二分的研究方法。在工程路线指导下，由于潜在目标是生产出符合某种性能目标的产品，因此就产生了面向性能的方法论；而在理论路线指导下，由于潜在目标是增进人类对计算智能的理解，因此关注重点是底层处理而非外在性能，结果就产生了面向模拟的方法论。

作为一个例子，考虑自然语言处理和语言学领域。这些领域关系密切，各领域的研究可互相取长补短，但它们的根本目标却不同。语言学家的兴趣在于弄明白人类如何处理语言，因此更倾向于理论性的研究，而自然语言处理领域的研究者的兴趣在于开发能处理自然语言的机器。所以，语言学家以面向模拟的模式运作，也就是建造用来检验理论的系统。相反，自然语言处理的研究者以面向性能的模式运作，也就是建造执行任务的系统。后一种模式产生的系统（如文档翻译机和响应口头命令的机器系统）在很大程度上依赖于语言学家获取的知识，但在特定系统的限定环境中经常使用可以工作的"简化方法"。

作为一个基本的例子，考虑为一个操作系统开发一个外壳（shell）的任务，它通过口述英语命令接收来自外部世界的指令。在这种情况下，shell（一个智能体）不需要考虑完整的英语语言。更准确地说，外壳不需要区分copy这个词的不同意思。（是名词还是动词？是否有抄袭的含义？）相反，外壳仅仅需要把copy这个词与其他命令（如rename、delete）区分开来就行。所以外壳只要将输入与预先确定的音频模式相匹配，就能够执行它的任务。这种系统的性能可能会令工程师满意，但从美学角度看，其实现方式也许并不能取悦理论家。

11.1.3　图灵测试

过去，**图灵测试**（Turing test，1950年由阿兰·图灵提出）一直作为衡量人工智能领域发展的一种标准。现在，图灵测试的重要性已不及从前，但它仍是人工智能领域重要的一部分。图灵的提议是，允许一个人（我们称他为询问者）与一个测试对象通过一个打字机系统进行通信，而没有告知询问者测试对象究竟是一个人还是一台机器。在这种环境中，如果询问者没能够把一台机器与一个人区分开来，那么可以说明这台机器的行为是智能的。图灵预测，到2000年，机器将会有30%的机会通过一个5分钟的图灵测试——这个预见惊人地准确！

人工智能的起源

寻求建造能够模仿人类行为的机器有很长的历史，不过很多人会认同现代人工智能领域起源于1950年。就在这一年，阿兰·图灵发表了论文"Computing Machinery and Intelligence"，提出机器能够通过编程来展现智能的行为。这个领域的名字——人工智能——就在几年后由约翰·麦卡锡（John McCarthy）在一个现在看来颇具传奇色彩的建议里提出，他建议"1956年夏天在达特茅斯学院（Dartmouth College）开展人工智能的研究"，以探究"这种推测，即认为认知的每个方面或智能的任何其他特征原则上都能够被精确地描述，从而能够制造出模拟它的机器"。

图灵测试已不再被认为是对智能的有效度量，其中一个原因在于，一个怪诞的智能显示可以用相对简单的手法**生产**（produce）出来。图灵测试场景的一个著名示例是20世纪60年代中期由Joseph Weizenbaum开发的程序DOCTOR（更通用的系统版本称为ELIZA）。这个交互程序被设计用来反映指导心理治疗的罗杰斯心理分析的场景，计算机扮演了分析师的角色，而用户扮演病人。在内部，DOCTOR所做的一切是根据某些定义明确的规则将病人的陈述重新构造并反馈给病人。例如，回应病人的陈述“我今天觉得很累”，DOCTOR可能回答：“为什么你今天觉得很累？”如果DOCTOR不能识别句子结构，它仅仅作出像“继续”或者“这很有趣”这样的响应。

Weizenbaum开发DOCTOR的目的是为了研究自然语言的交流。心理疗法的目标仅仅提供了一个程序可以“交流”的环境。然而，Weizenbaum没有想到的是，个别的心理学家建议把这个程序用在实际的心理治疗中。（罗杰斯的论点是，在治疗期间，应该是病人来主导对话，而不是分析师。所以他们认为计算机也能像治疗师那样引导对话。）而且，DOCTOR表现出极强的理解能力，以至许多与它“沟通”过的人会迎合这种与机器的问答式对话。在某种意义上，DOCTOR通过了图灵测试。其结果是，在伦理及技术上都产生了争议，Weizenbaum成为一位在技术进步的世界中维护人类尊严的倡导者。

较新的图灵测试“成功”的例子包括一些因特网病毒，为了诱骗人类放松对恶意软件的防护，它与人类受害者进行“智能”对话。此外，与图灵测试类似的现象发生在下象棋等计算机游戏的场景中。尽管这些程序选择棋路时仅仅通过应用蛮力技术（这与我们将在11.3节讨论的内容相似），但人类在同计算机的竞赛过程中常感觉机器拥有创造力甚至个性。相似的感觉发生在机器人技术领域，根据物理属性建造的机器表现出了智能的特征。例如，玩具机器狗仅仅通过点头或者是竖耳朵来响应声音，就表现出了可爱的个性。

问题与练习

1. 指出一个智能体可能会做的几种“智能”动作。
2. 一棵放在只有一束光源的暗室里的植物，会朝着光源方向生长。这是一种智能响应吗？植物拥有智能吗？你对智能的定义是什么？
3. 假定一台售货机根据所按的按钮发售不同的物品，你认为这样的一台机器是否“知道”按了哪个按钮？你对“知道”的定义是什么？
4. 如果一台机器通过了图灵测试，你会认同这台机器是智能的吗？如果不是，你是否认同该机器看上去是智能的？
5. 假设你使用聊天室通过因特网与某人聊天（或使用某种即时通信软件，如Instant Messenger），并与之进行了长达10分钟的意味深长的连贯谈话。如果你在稍后发现之前是与一台机器对话，你是否会认为这台机器是智能的？为什么？

11.2 感知

一个智能体要智能地响应从它的传感器接收的输入，就必须能够理解输入。也就是说，智能体必须能够感知。本节我们来探讨感知的两个研究领域，即理解图像和理解语言，这两个领域被证明特别具有挑战性。

11.2.1 理解图像

让我们考虑11.1节介绍的解决八数码游戏的机器所提出的问题。机器上夹具的张合没有表

现出严重的障碍，在这个张合的过程中，因为这里的应用要求的精度不高，所以检测夹具中是否有拼图的功能也很简单，即使是摄像机对拼图的对焦问题也不难解决，方法是通过设计夹具把拼图安置在一个事先确定的位置，以便观察。所以，机器需要的第一个智能行为是从视觉媒介中提取信息。

必须认识到，机器在看拼图时所面对的问题不单是产生和存储一张图像，这方面的技术很多年前在传统的摄影及电视系统中就能做到。相反，这里的问题是为了提取拼图当前的状态，要理解这个图像（可能随后还要监控方块的移动）。

在八数码游戏机器的案例中，对拼图图像的可能解释是相对有限的。我们可以假定，在一个排列好的模式里所呈现的总是一幅包含数字1~8的图像。问题只是去提取这些数字的排列。为此，我们想象拼图的图像已经在计算机内存中，按照位进行编码。编码中的每一位表示具体像素的亮度。假定图像的大小统一（机器把拼图放在摄像机前的预定位置），把图像的不同部分与由位模式构成的预定模板相比较（位模式是由拼图中使用的单个数字生成的），机器就能够检测出哪个方块在哪个位置。如果发现匹配，则说明拼图达到了要求的条件。

这种图像识别技术是光学字符阅读器中使用的一种方法。但它是有缺点的，它对被读的符号在类型、大小以及方位上要求一定程度的一致性。特别是，即使对于相同的符号，外形也相同，但字体较大的字符产生的位模式与较小字体的模板也不匹配。此外，可以想象，当试图处理手写材料时，问题会变得非常困难。

解决字符识别问题的另一个方法基于匹配几何特征，而不是符号的精确外形。在这种情况下，数字1表示为一条单竖线，数字2可能代表一条不封闭的曲线，底部与一条水平直线相连，等等。这种符号识别的方法分两步：第一步是从要处理的图像中提取图像特征，第二步是把这些特征与已知符号的特征进行比较。与模式匹配方法一样，这种符号识别技术并不可靠。例如，图像的少许错误会产生一组完全不同的几何特征，比如区分字母O和C，以及八数码游戏里的数字3和8时就容易混淆。

幸好在八数码游戏中不需要理解一般的三维图像。例如，我们的优势是能保证识别的形状（数字1~8）相互孤立地处在图像上的不同部分，而不是常见的重叠图像。例如，在一张普通的照片中，我们面临的不仅是从不同的角度识别一个对象的问题，还包括被遮蔽的对象的某些部分。

理解一般图像的任务通常采取两步进行处理：（1）**图像处理**（image processing），指标识图像的特征；（2）**图像分析**（image analysis），指理解这些特征代表什么意思的过程。在利用符号的几何特征识别符号的描述中，我们已经提出了二分的处理方法。在这个过程中，图像处理由标识在图像中发现的几何特征的过程来表示，图像分析由标识那些特征的含义的过程来表示。

图像处理带来了大量的研究课题。一个是轮廓增强，使用数学技术使图像中区域间的边界线变得更清晰。在某种意义上，轮廓增强试图将照片转换成线条画。图像分析的另一个活动是区域查找。这是标识图像中区域的过程，这些区域拥有共同的属性，比如亮度、颜色或者纹理。这样的一个区域很可能代表图像中某个对象的一部分。（这种识别区域的能力使得计算机可以给老式的黑白电影着上彩色。）图像处理领域还包含另一个活动——平滑（Smoothing），即去除图像中的缺陷。平滑使图像中存在的错误不会混淆其他图像处理步骤，但是过度平滑会导致重要信息的丢失。

平滑、轮廓增强以及区域发现都是用来标识图像中各种成分的步骤。图像分析是确定这些成分代表什么，以及最终确定这个图像代表什么的过程。这里我们还要面对从不同视角识别被部分遮挡的对象这样的问题。图像分析的一种方法是首先假定一个图像大概是什么，然后尝试把图像中的成分与那些猜测的对象相联系。这看起来是人类所使用的方法。例如，有时我们会

发现，在我们视觉模糊的情况下识别一个没有想到的对象是很困难的，但是一旦有一个该对象可能是什么的线索，我们就能容易地认出它。

与一般图像分析相关的问题特别多，在这个领域还有许多研究要做。图像分析这个领域证明，那些人类智能能够很快又非常容易完成的任务，却对机器能力提出了强有力的挑战。

强人工智能与弱人工智能

能够通过对机器进行编程来展现其智能行为的那种推测能力被认为是**弱人工智能**（weak AI），今天在不同程度上已经被大众所接受。但是，机器能够通过编程而获得智力，亦即意识的那种推测能力，则被认为是**强人工智能**（strong AI）。强人工智能引发了广泛的争论。反对者认为，机器在本质上与人类不同，它永远不能像人类那样感受爱、判断对错，以及考虑自我。然而，支持者认为，人类的头脑是由许多小的部件构成，每个部件都不是人，没有意识，但是当它们结合在一起就成了人。他们辩称，为什么同样的现象就不可能出现在机器身上呢？

解决强人工智能争辩的难点在于，智能和意识这样的属性是内在特性，不能够直接界定。正如阿兰·图灵指出的那样，我们认为其他人属于有智能的是因为他们的行为表现出智能——即使我们不能观察到它们内部的智力状态。那么，如果机器也呈现外在的意识特性，我们是否准备认可机器具备同样的水准呢？为什么是，为什么不是？

11.2.2 语言处理

理解语言是感知问题的另一个已证明的颇具挑战性的问题。把形式化的高级程序设计语言翻译成机器语言（参见6.4节）获得的成功使早期的研究人员认为，通过编程使计算机具有理解自然语言的能力将在几年后变成现实。实际上，翻译程序的这种能力给我们一种错觉——机器真能理解被翻译的语言。（回忆6.1节中Grace Hopper讲的故事，那个经理以为她正在教计算机理解德语。）

这些研究人员没有明白形式化的程序设计语言与英语、德语以及拉丁语这些自然语言之间在深度上的差异。程序设计语言由精心设计的原语组成，每个语句只有一种语法结构，只有一种意思。相反，自然语言的一个语句会因为上下文的不同，甚至交流方式不同而有多种意思。因此，人类理解自然语言很大程度上依靠额外的知识。

例如，句子

Norman Rockwell painted people.

以及

Cinderella had a ball.

都有多种意思，但通过语法分析或单独翻译每个词并不能区分这些意思。实际上，要理解这些句子需要有理解句子上下文的能力。在有些场合，一个句子的真实意思与它的字面意思完全不同。例如，“你知道几点了吗？”通常的意思是：“请告诉我现在几点了。”或者如果说话者已经等候了很长时间，那么这句话意思可能是：“你来得太迟了。”

要弄明白一种自然语言中的一个句子的意思需要几个层次的分析。第一层是**语法分析**（syntactic analysis），其主要成分是句法分析。在这里，句子

Mary gave John a birthday card.（玛丽给约翰一张生日贺卡。）

的主语是Mary，而句子

John got a birthday card from Mary.（约翰收到玛丽赠送的一张生日贺卡。）

的主语是John。

分析的第二层次称为**语义分析**（semantic analysis）。语法分析仅标识每个词在语法上的作用，与语法分析不同，语义分析的任务是标识句子中的每个词在语义上的作用。语义分析试图标识这样的内容，如描述的动作、动作的主体（可能是句子的主语，也可能不是）以及动作的目标。正是通过语义分析，我们认为句子“玛丽给约翰一张生日贺卡”和“约翰收到玛丽赠送的一张生日贺卡”是在说同一件事情。

分析的第三层是**上下文分析**（contextual analysis）。在这一层，句子的上下文被引入理解过程中。例如，很容易分辨出句子

The bat fell to the ground.

中的每一个单词在语法上的作用。我们甚至能通过识别动作“falling”和动作主体“bat”等来实现语义分析。但只有等我们考虑到上下文后，句子的意思才能变得明确。尤其是，这句话在棒球比赛这样的背景下和在洞穴中探险这样的背景下有着不同的意思[①]。而且，只有在上下文这一层，问题“你知道几点了?”的真正意思才能最终揭晓。

我们应当注意，各个层次分析（语法、语义及上下文）并不一定相互独立。对于句子

Stampeding cattle can be dangerous.

如果我们想象是一群牛在惊跑，那么主语是名词cattle（用形容词stampeding加以修饰）。但是如果语境是哪个恶作剧者以惊吓牛群取乐，那么主语就是动名词stampeding（宾语是cattle）。因此，这个句子不只有一个语法结构——究竟哪一个意思正确要依赖于上下文。

自然语言处理的另一个研究领域涉及整个文档，而不是单个句子。这里涉及的问题可分成两类：**信息检索**（information retrieval）和**信息提取**（information extraction）。信息检索的任务是标识与手头论题有关的文档。例如万维网的用户试图找到与特定主题相关的站点时所面对的问题。该技术的当前状态是为关键字搜索站点，但这经常产生大量的错误链接，并且由于其处理“automobiles”而不是“cars”，它会忽略一个重要的站点。需要的就是能理解所考虑站点内容的搜索机制。获取这样的理解是很困难的，这也就是许多搜索机制都转向采用像在4.3节介绍的XML这样的技术，来产生语义网Web的原因。

信息提取是指从文档中提取信息这样的任务，并采用一种形式以方便用于其他应用程序。这个意思可以理解为为一个特定的问题确定答案，或者是将信息以某种格式记录，以备日后解答问题时使用。有一种这样的格式称作**框架**（frame），框架实质上是一种记录细节的模板。例如，考虑一个读报系统，该系统可以利用各种各样的框架，报纸上的每一类文章用一个。如果系统认定一篇文章是关于入室盗窃的报道，它会继续试图把这填写到入室盗窃框架中，该框架可能要求填写这样一些条目，如失窃地点、失窃时间和日期以及失窃物品等。相反，如果系统认定一篇文章是关于自然灾害的报道，那么它就填写自然灾害框架，引导系统确定灾害类型、损失的大小等。

信息提取者记录信息的另一种形式称为**语义网**（semantic net）。这实质上是一个大的链式数据结构，结构中的指针用来指示数据项之间的联系。图11-3显示了一个语义网的一部分，突出

---

① 英文单词bat有蝙蝠和球棒两种意思，在棒球比赛中应指“球棒”，在洞穴中探险时则可能指“蝙蝠”。
——译者注

显示的部分是从句子

Mary hit John.

中得到的信息。

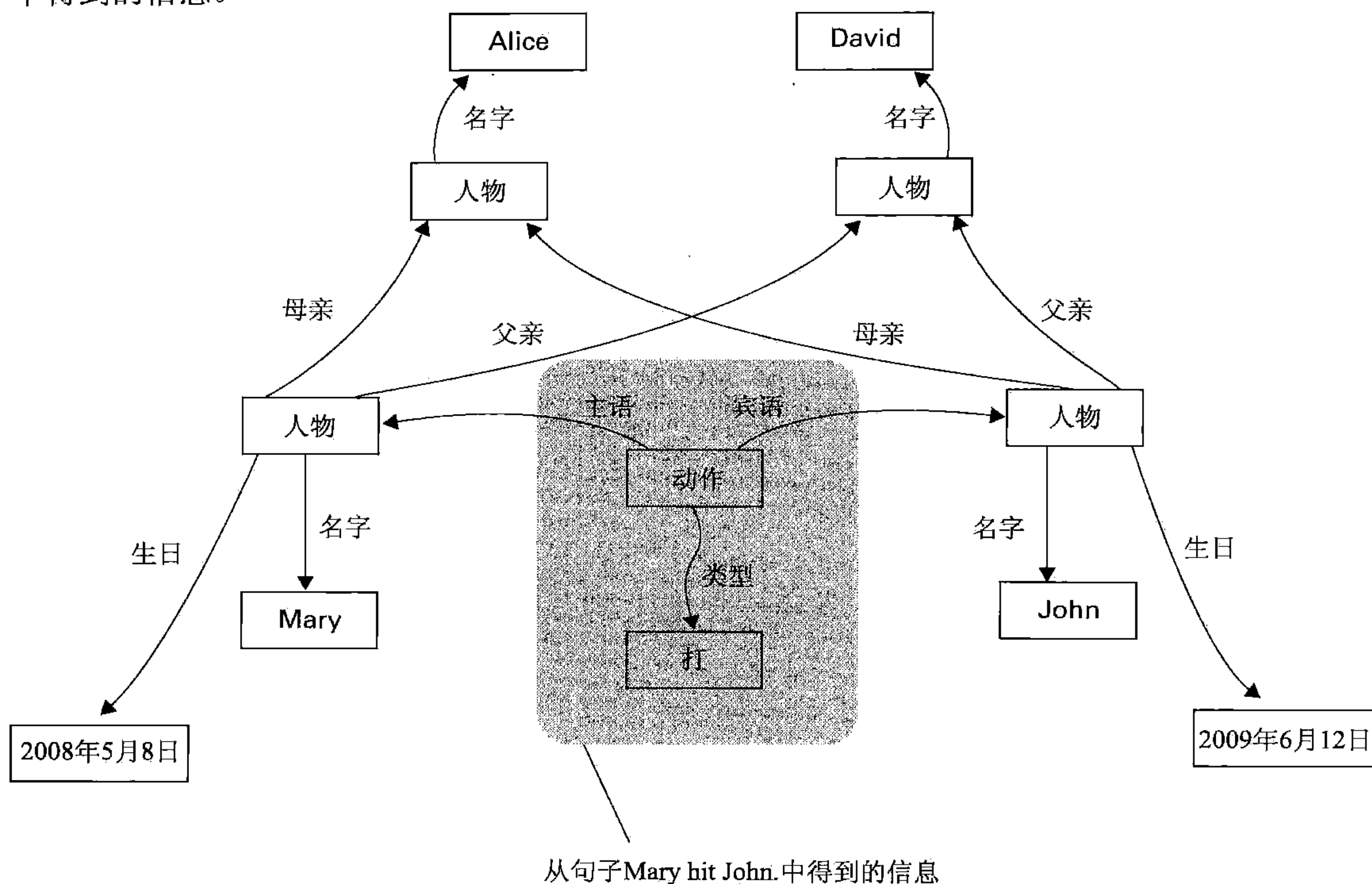


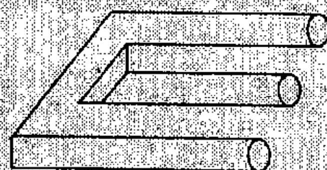
图11-3 一个语义网

掌中的人工智能

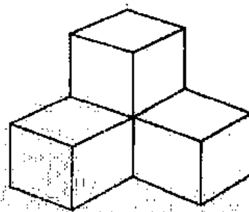
人工智能技术正在智能手机应用中越来越多地显现出来。例如，谷歌研发了Google Goggles——一个提供虚拟搜索引擎的智能手机应用。只要用智能手机的照相机对某一本书、某一地标性建筑或某一标记拍照，Goggles就会执行图像处理、图像分析和文本识别，然后启动一次Web搜索来识别对象。如果讲英语的你正身处法国，那么可以为一个标记、菜单或其他文本拍照，然后把它翻译成英文。除Goggles外，谷歌正在积极地研究声音对声音的语言翻译，很快你就可以用英语对手机讲话，然后让手机播放其对应的西班牙语、中文或其他语言的声音。随着不断以创新的方式使用AI，智能手机无疑会越来越智能。

问题与练习

1. 对于一个机器人视频系统，一种情况是机器人用该系统来操控自己的活动，另一种情况是将其传递给遥控机器人的人。这两种情况对视频系统的要求有什么不同？
2. 是什么让你知道下图不合情理？这样的洞察力怎样编程到机器中？



3. 下图中有几个方块？怎样对一个机器编程使其能正确回答这个问题？



4. 你是如何知道句子“Nothing is better than complete happiness”和句子“A bowl of cold soup is better than nothing”不是暗指“A bowl of cold soup is better than complete happiness”?你这种区分能力如何被移植到一台机器上？
5. 指出在翻译句子“They are racing horses”时的二义性。
6. 比较下列两个句子的语法分析结果。然后解释二者在语义上的不同。

   The farmer built the fence in the field.

   The farmer built the fence in the winter.
7. 根据图11-3中的语义网，Mary和John之间是什么家庭关系？

11.3 推理

现在，让我们来利用11.1节介绍的求解八数码游戏的机器来探讨开发具有基本推理能力的智能体的技术。

11.3.1 产生式系统

解决八数码游戏的机器从看到的图像中解读出了方块的位置以后，它的任务就变成了决定需要哪些移动来求解难题。马上可以想到的一个方法是，把方块的所有可能排列对应的解决方案都预先编写到机器中。然后，机器的任务就只是选择和执行合适的程序。然而，这个八数码游戏有100 000多种布局，所以对每一种布局提供一个直接的解决方法的方案显然不可取。因此，我们的目标是对机器编程，让机器能够自己构建难题的解决方法。也就是说，必须对机器编程使其能够实现基本的推理活动。

开发机器的推理能力已经是一个研究多年的主题。有关这方面的研究已经达成一个共识，即有一大类推理问题具有共性，这些共性被单独放在一个抽象的实体中，该实体称为**产生式系统**（production system）。这种系统由三个主要部分组成。

(1) 状态集合。每个**状态**（state）是一个可能在应用环境中发生的情形。最初的状态称作**开始状态**（start state）（或者初始状态），期望的状态称作**目标状态**（goal state）。（在我们的案例中，一个状态就是指八数码游戏的一种布局，开始状态就是八数码游戏提交时的布局，目标状态就是图11-1所示的已经解决了难题的布局。）

(2) 产生式集合（又称规则或者移动）。**产生式**（production）是指能在应用环境中执行的一个操作，以使系统从一个状态转移到另一个状态。每个产生式可以与一些先决条件相关联，也就是说，在应用产生式之前，环境中必定会出现一些可能存在的条件。（在我们的案例中，产生式就是方块的移动。一个方块每次移动的前提条件是其相邻的位置必须有空位。）

(3) 控制系统。**控制系统**（control system）是由解决问题使其从开始状态变换到目标状态的逻辑组成的。在处理过程的每一步，控制系统都要决定，在满足先决条件的那些产生式中，下一步该执行哪一个。（对于八数码游戏的例子，给定一个特定状态，在空位旁会有几个方块，因而存在几个可用的产生式。控制系统必须决定移动哪一个方块。）

注意，在一个产生式系统的上下文中，赋予解决八数码游戏的机器的任务可以被公式化。在这种情况下，控制系统采用程序的形式。该程序检查八数码游戏的当前状态，确定实现目标状态的一系列产生式，并执行这一系列产生式。因此，我们的任务就是为解决八数码游戏设计一个控制系统。

控制系统开发中的一个重要概念是**问题空间**（problem space），它是产生式系统中的所有状态、产生式以及先决条件的集合。问题空间通常概念化成**状态图**（stategraph）的形式。这里，图这个词是指一种数学家称为**有向图**（directed graph）的结构，是指一组由箭头连接起来的称为**节点**（node）的位置。一个状态图由一组用箭头连接的节点组成，节点表示系统中的状态，箭头表示从一个状态转换到另一个状态的产生式。状态图中两个节点被一个箭头连接的条件是：当且仅当有一个产生式，它把系统从箭头起点处的状态转换到箭头终端处的状态。

我们应当强调的是，正像在解决八数码游戏时拼图可能状态的数量使我们难以明确地提供预先编写好的解决方案一样，数量太大的问题也使得我们不能明确地表示整个状态图。所以，状态图是概念化手头问题的一种方法，但不能用来表示全部内容。虽然如此，你会发现，考虑图11-4显示的八数码游戏的一小部分状态图对于求解难题是有帮助的。

当根据状态图来考虑时，控制系统面临的问题变成了寻找一连串从开始状态导向目标状态的箭头。毕竟，这一连串的箭头代表了解决初始问题的一系列产生式。所以，不管应用是什么，控制系统的任务都可以看做是寻找一条贯穿状态图的路径。对控制系统的这种普遍观点是根据产生式系统对要求推理的问题进行分析的成果。如果一个问题能够根据产生式系统来描绘，那么它的解决方法就能够根据搜索一条路径来明确表达。

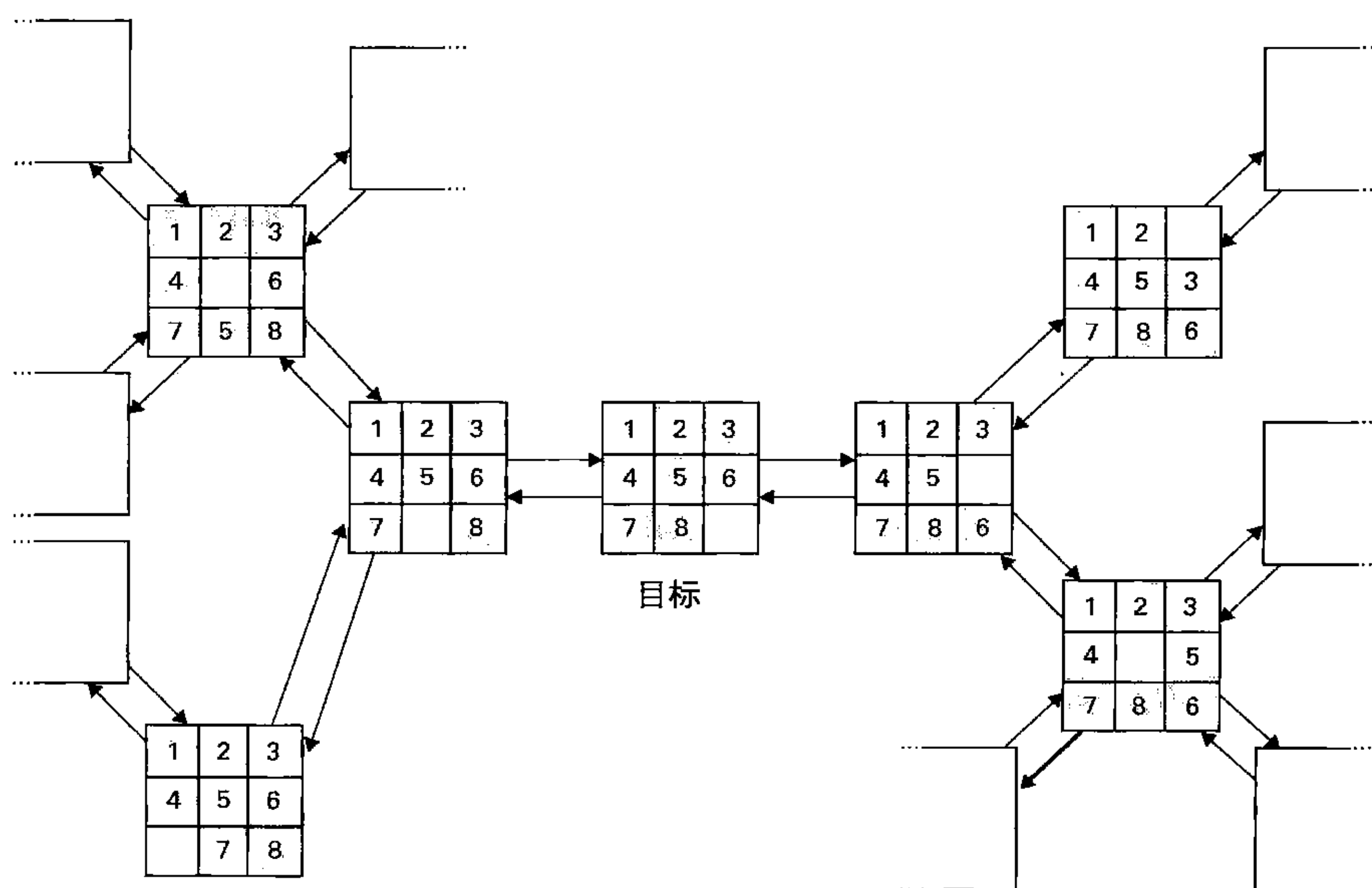


图11-4 八数码游戏状态图的一小部分

为了强调这一点，我们先来考虑其他任务是如何按照产生式系统来设计，然后在控制系统通过状态图发现路径的背景下完成的。人工智能的经典问题之一就是下象棋这样的游戏，这类游戏在一个明确规定的背景下属于中等复杂度，因此，它为理论测试提供了一个理想的环境。在象棋游戏中，基本产生式系统的状态是可能的棋盘布局，产生式是棋子的移动，控制系统就具体为棋手（人或别的）。状态图的开始节点表示棋子在初始位置时的棋盘。从该节点出发的分

支是一些箭头，这些箭头指向游戏中棋子第一步移动之后会形成的那些棋盘布局，而从这些节点出发的每一个分支又指向下一步移动棋子会形成的那些布局，依次类推。通过这种明确的表达，我们可以把一个象棋游戏想象为由两个选手组成，每个选手都试图通过在一个大的状态图中寻找一条通向自己选择的目标节点的路径。

或许从给定事实得出逻辑推论的问题是一个不太明显的产生式系统的例子。在这种情况下，产生式是称为**推理法则**（inference rule）的逻辑规则，这些规则允许从旧命题中形成新命题。例如，命题“所有超级英雄都是崇高的”和“超人是超级英雄”可以合并产生“超人是崇高的”。这样一个系统中的状态由各种命题组成，这些命题在推导过程的一些特定点为真：开始状态是基本命题（常称为公理）的集合，从中可以得出结论；而目标状态则是包含了所提出结论的命题的任意集合。

例如，图11-5显示了以下推论所经历的状态图的一部分。从一组命题“苏格拉底是男人”、“所有男人都是人”及“所有人都终有一死”可以推出“苏格拉底终有一死”。从中我们看到，随着推理过程应用合适的产生式来生成新的命题，知识的主体从一个状态转换到了另一个状态。

当今，这种推理系统经常应用在逻辑程序设计语言（6.7节）中，它是大多数**专家系统**（expert systems）的核心。专家系统是为模拟因果推理而设计的软件包，当人类专家面对相同的情形时将遵从这些因果推理。例如，医疗专家系统用来协助疾病诊断或改良治疗。

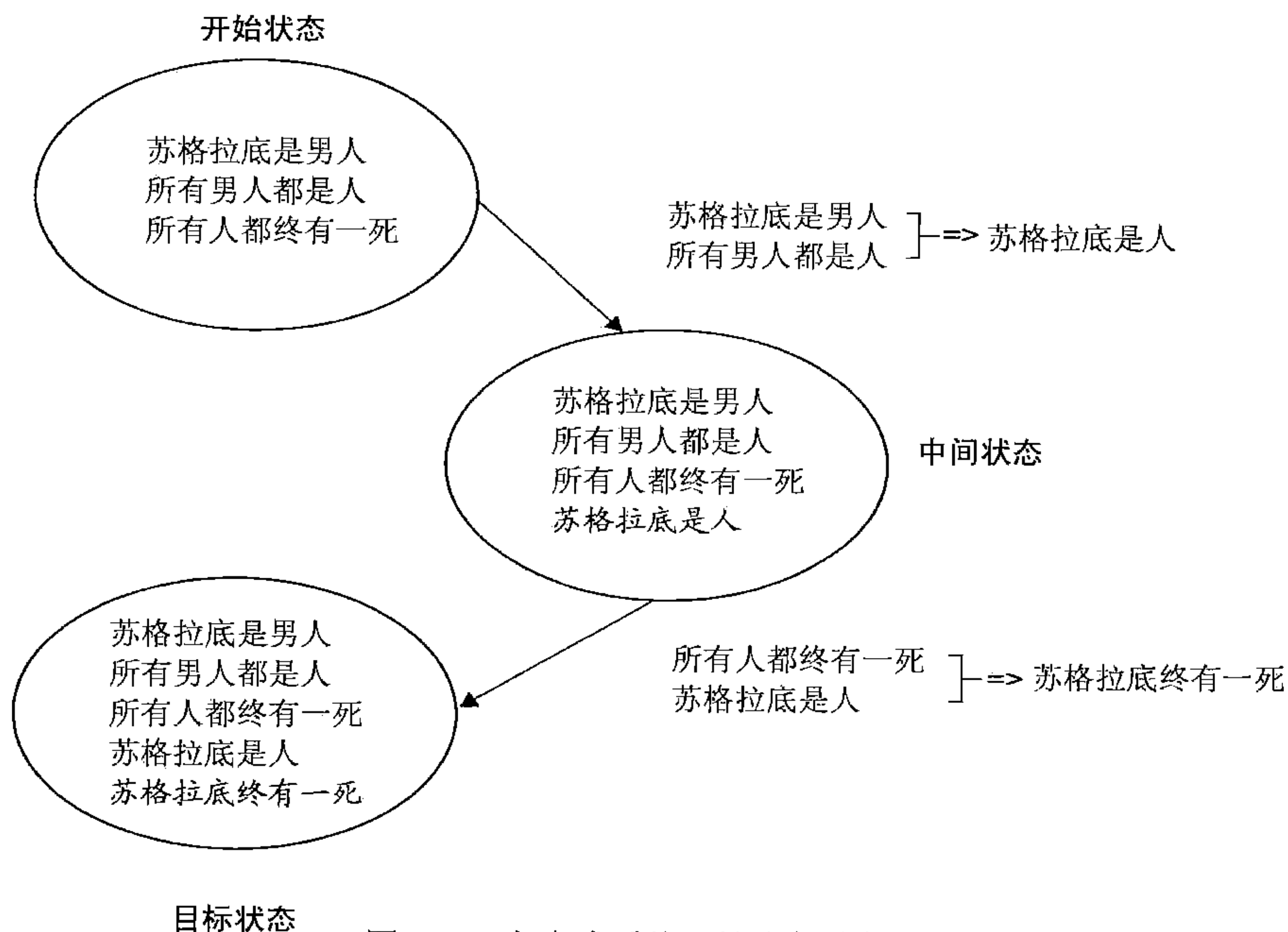


图11-5　产生式系统环境中的演绎推理

11.3.2　搜索树

我们已经看到，在产生式系统的环境中，控制系统的工作涉及搜索状态图，找出从开始节点到目标节点的一条路径。完成这种搜索的一个简单的方法就是仔细观察每一个从初始状态发出的箭头，并记录下每一个目标状态，然后继续仔细观察从这些新状态发出的箭头，再记录下结果，依次类推。对目标的搜索像向水中滴入一滴墨水一样，从开始状态扩散开来。这个过程继续进行，直到一个新状态就是目标状态，在这里，解决方法就找到了，控制系统只需要沿着找到的这条从开始状态到目标状态的路径应用产生式即可。

这种策略的结果实际上是建立一棵树，称作**搜索树**（search tree），它由被控制系统分析后得到的部分状态图构成。搜索树的根节点是开始状态，每个节点的子节点由那些应用一个产生式从父节点可到达的状态构成。搜索树中节点间的每条线段代表一个产生式的应用，从根到叶子的每一条路径代表状态图中相应状态间的一条路径。

解决图11-6所示布局的八数码游戏将会产生的搜索树如图11-7所示。该搜索树最左边的分支代表试图通过最初向上移动方块6来解决问题，中间分支表示向右移动方块2的开局方法，而最右边的分支表示以向下移动方块5来开局。搜索树进一步显示，如果以向上移动方块6来开局，那么下一个允许的产生式只能是方块8右移。（实际上，这时还可以将方块6向下移动，但这仅仅是上一个产生式的倒置，因而是毫无意义的移动。）

1	3	5
4	2	
7	8	6

图11-6 一个尚未解决的八数码游戏

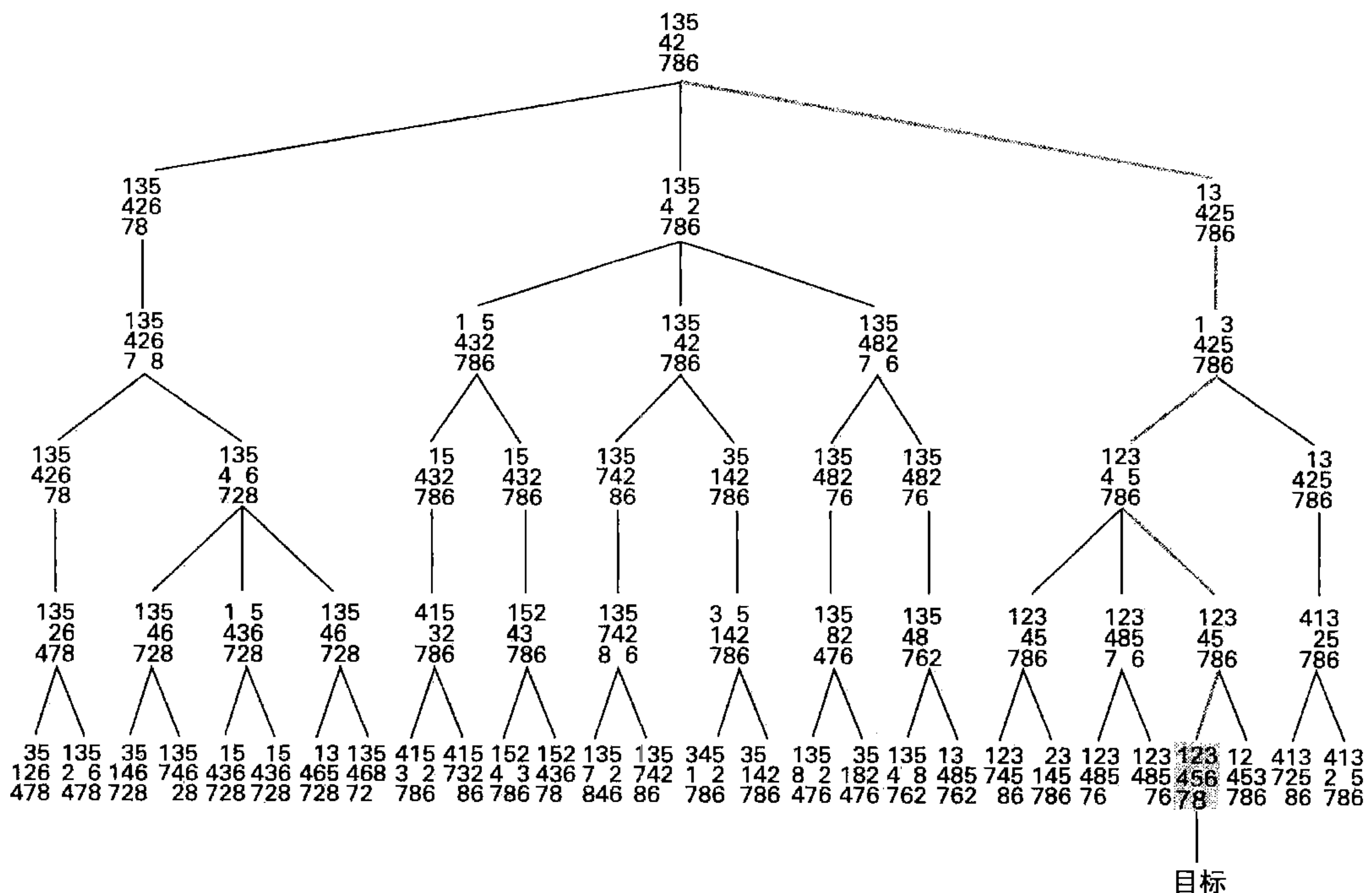


图11-7 搜索树的一个示例

目标状态出现在图11-7显示的搜索树的最下面一层。因为这预示了已经找到了一个解决方法，所以控制系统可以结束搜索过程并开始构建指令序列，该指令序列将用来解决外部环境中的拼图难题。实际上是一个简单的过程：从目标节点的位置上行，同时在遇到由树枝线表示的产生式时将它们压入栈。将这种技术应用到图11-7所示的搜索树中，产生了如图11-8中所示的产生式栈。现在，只要把指令从该栈中弹出并执行，控制系统就能够解决外界的问题。

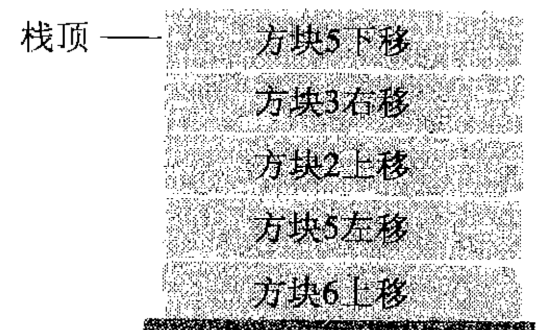


图11-8 压入栈的产生式以备以后执行

还有一点我们应当注意，回想第8章我们讨论的树，利用一个指针系统来指向下面的树，由此可以从一个父节点移动到它的子节点。但是在搜索树的情况下，控制系统必须能够从一个子节点移到其父节点，正如从目标状态向上移到开始状态。构建这种树的指针系统要向上指，而不是向下指。也就是说，每个子节点包含了一个指向其父节点的指针，而不是父节点包含了指向其子节点的指针。（有些应用会用到这两组指针，允许在树中双向移动。）

11.3.3　启发式搜索

对于图11-7显示的例子，我们选择一个开始布局，产生了一个容易处理的搜索树。但是在试图解决一个比较复杂的问题时，产生的搜索树就会变得非常庞大。在国际象棋中，第一步移动就有20种可能，因此在这样的情况下，搜索树的根节点将会有20个子节点，而不是我们例子中的3个。而且，一局棋双方都移动30~35次是常有的情况。即使是八数码游戏，若不能很快到达目标，搜索树也会变得非常大。结果，开发一个完整搜索树同表示出全部状态图一样都是不切实际的。

应对这种问题的一种策略是改变搜索树构建的次序。不用**广度优先**（breadth-first）的方式（这意味着树是一层一层地构建），我们可以沿着更有希望的路径往深度发展，但只有在原来的选择失败以后才考虑其他选择。结果就是以**深度优先**（depth-first）的方法构建搜索树，也就是说，树是以建立纵向路径而不是横向层次的方式构建的。更确切地说，这种方法通常被称为**最佳优先**（best-first）结构，因为在搜索中被选中的垂直路径看起来是最优的。

最佳优先的方法类似于人类面对八数码游戏时应用的策略。我们一般不会像广度优先方法那样，同时沿着几个可能的路径进行。相反，我们大概会选择看起来最有希望的路径并首先沿着这条路径走下去。注意，我们说的是看上去最有希望的。在一个特定点，很难确定哪个选择是最佳的。只凭感觉，当然可能“误入歧途”。但不管怎样，相比“一视同仁”地关注每种选择的“笨方法”，这种凭直觉的方法似乎更具优势，所以在自动控制系统中应用直觉的方法似乎是明智的。

为此，我们需要一个方法来确定几个状态中哪一个看上去是最有希望的。我们的方法是采用**启发式**（heuristic），在本例中这是与每个状态相关的一个数值，用来衡量这个状态与最近目标之间的“距离”。在某种意义上，启发值是衡量规划代价的一个尺度。给定两个状态之间的一个选择，那么从具有较小启发值的状态到达目标，显然花的代价更小。因此，该状态代表了应遵循的方向。

启发值应具备两个特征。第一，如果到达相应的状态，它必须对该解决方案中剩余的工作量有一个合理的估计。这意味着它在多个选项中作出选择时能提供有意义的信息——启发值提供的估计越精确，根据此信息所做的决定就越正确。第二，启发值应容易计算。这意味着它的利用应有益于搜索过程而非成为一种负担。如果计算启发值非常复杂，那倒不如把时间花费在推导一个广度优先树上。

在八数码游戏中，一个简单的启发式搜索要通过计算不在合适位置上的方块数目来估计到达目标的“距离”——这种推测指的就是一个有4个方块不在合适位置上的状态，相对于只有2个方块不在合适位置上的状态来说离目标更远（也就因此更缺少吸引力）。然而，启发值并没有考虑方块离其位置有多远。假如这两个方块离它们的正确位置太远，就需要许多产生式来移动它们。

于是，一个比较好的启发式搜索测算每个方块离其终点的距离，并把这些值相加得到一个量。一个直接挨着其终点的方块的距离值为1，而有一个角与其终点方块相接触的方块的距离值为2（因为它至少要在垂直方向和水平方向各移动一次）。这种启发值容易计算，并对拼图从当

前状态到目的状态过程中需要移动的步数有一个粗略的估计。例如，图11-9所示布局相应的启发值为7（因为方块2、5和8与其终点的距离都为1，而方块3和6与终点的距离都为2）。实际上，它的确需要移动7步来完成拼图。

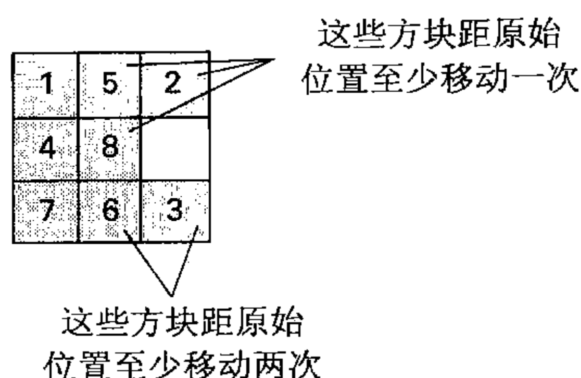


图11-9　一个未解的八数码游戏

既然我们有了八数码游戏的一个启发值，下一步就把它结合进决策过程。一个人在做决定的时候倾向于选择看起来更接近目标的选项，还记得吗？所以我们的搜索过程应当考虑树中每个叶子节点的启发，并且从启发值最小的叶子节点进行搜索。这就是图11-10所采用的搜索策略。图中给出了开发一个搜索树并执行得到的解决方法的算法。

让我们把这个算法应用到八数码游戏，从图11-6给出的初始布局开始。首先，我们建立初始状态并将它作为根节点，记录下它的启发值5。然后，如图11-11所示，while循环体的第一次循环添加了3个从初始状态能达到的节点。注意，我们已经在节点下的括号里记录了每一个叶子节点的启发值。

```
创建状态图的开始节点作为搜索树的根节点，并记
  录它的启发值
while (目标节点还没有到达) do
  [选择所有叶子节点中有最小启发值的最左边的
     叶节点
   将这个选定的节点作为子节点附加到通过单个
     产生式能到达的节点
  在搜索树中节点的旁边记录每一个新节点的启
     发值]
从目标节点向上遍历搜索树，把与每个经过的线段
  相关联的产生式压入栈
通过执行从栈中弹出的产生式解决原始问题
```

图11-10　采用启发值的控制系统的一个算法

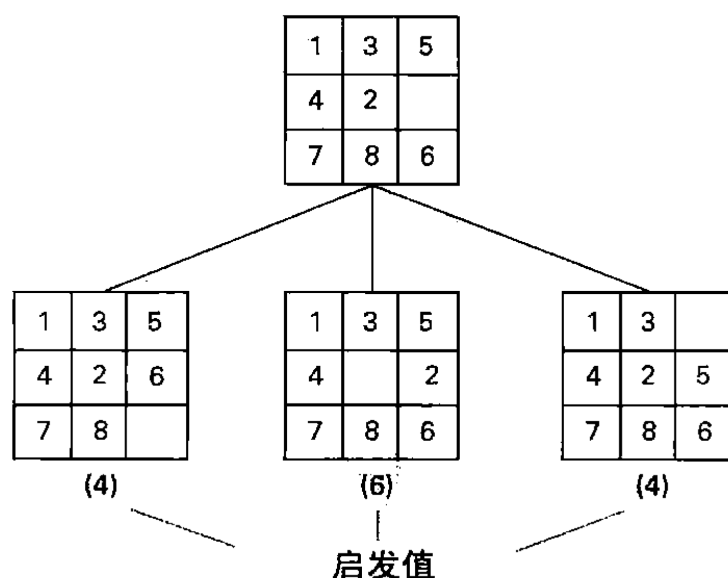


图11-11　启发式搜索的开始

目标还没有达到，所以我们再次执行while循环体，这次是从最左边的节点扩展搜索（“有最小启发值的最左边的叶子节点”）。结果搜索树呈图11-12所示的形式。

现在，最左边叶子节点的启发值是5，说明这个分支也许根本不是一个好选择。算法注意到这一点，在下一次循环时，while语句会指示从最右边节点的扩展搜索树（它现在是“有最小启发值的最左边叶子节点”）。这样扩展后的搜索树如图11-13所示。

这时，算法好像走上正轨。因为最右边节点的启发值是3，while语句指示继续沿着这条路径进行，搜索直瞄目标，产生了如图11-14所示的搜索树。这个树同图11-7所示的搜索树相比较表明，新算法即使早期走了点弯路，但启发信息的利用已经大大减少了搜索树的大小，并且处理效率大大增加。

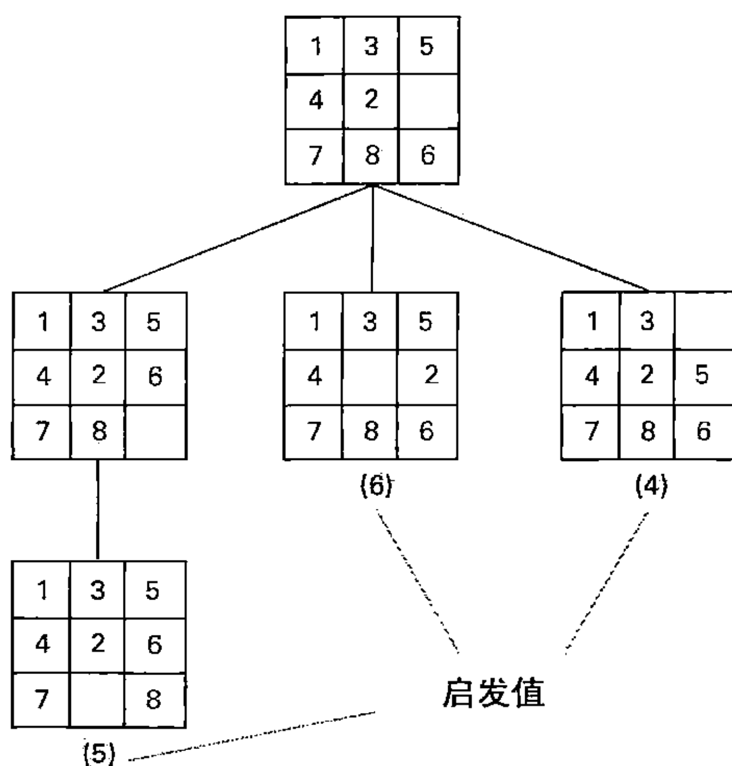


图11-12 2次搜索后的搜索树

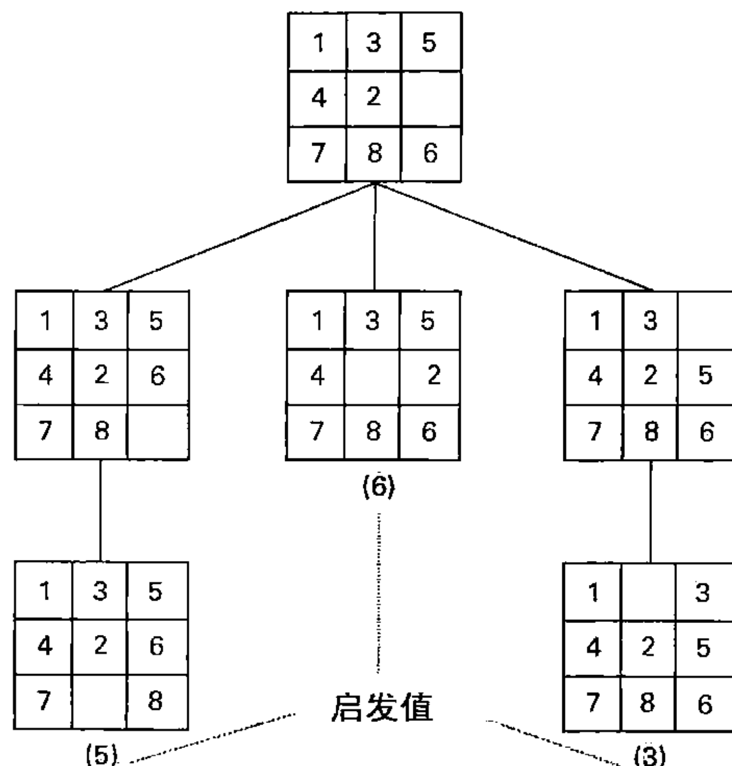


图11-13 3次搜索后的搜索树

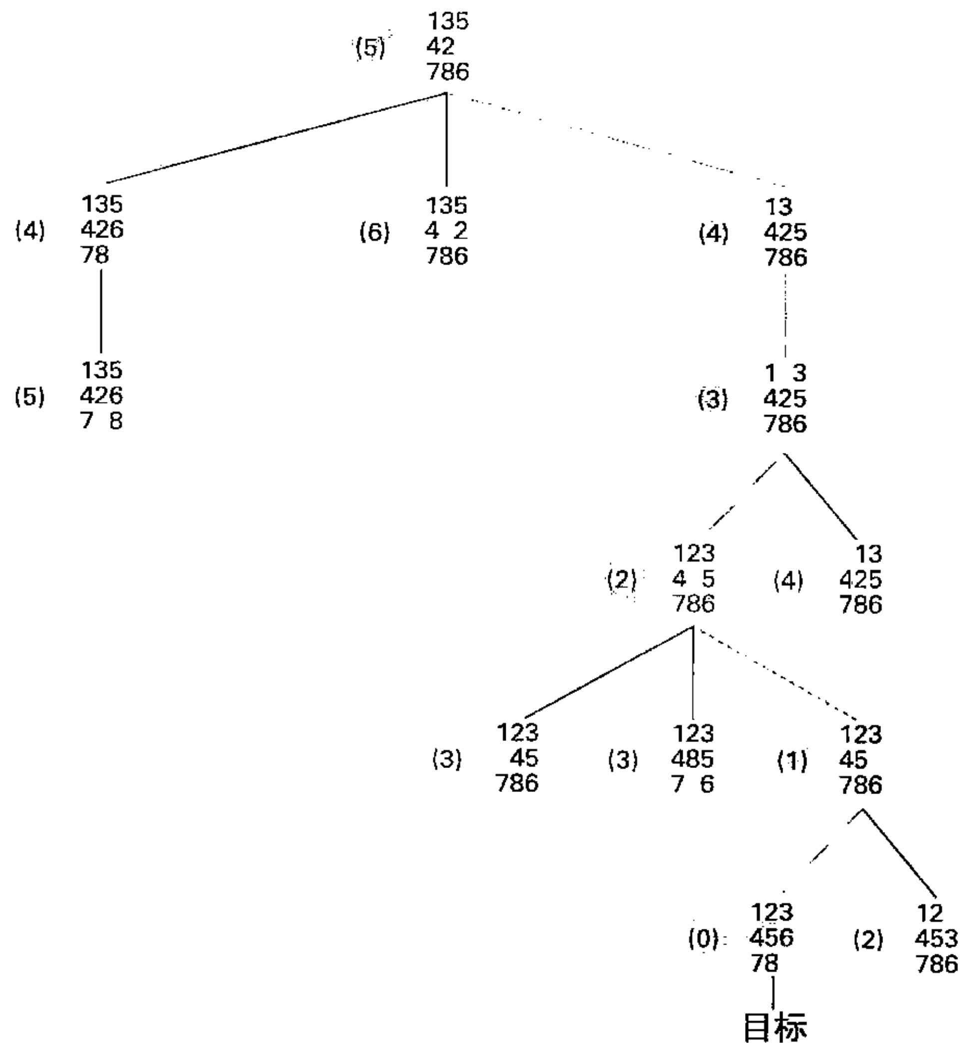


图11-14 用启发系统形成的完整搜索树

在到达目标状态之后，`while`语句终止，我们从目标节点反向向上移动到根节点，沿途把遇到的产生式压入一个栈。结果产生的栈就像前面描述的那样，如图11-8所示。

最后，当这些产生式从栈中弹出时，我们得到指示，执行它们。这时，我们可以看到解决拼图难题的机器放下它的指杆，开始移动方块。

最后，关于启发式搜索还要补充一点。我们在这一节中推荐的算法通常称为最佳适应算法，但它并不能保证是所有应用的最佳解决方案。例如，当使用汽车的GPS寻找到达某一城市的路线时，人们通常希望找到最近的路线，而不是任意一条路。**A*算法**（A* algorithm，读作“A星算法”）是最佳适应算法的修改版，它可找到一个最佳方案。这两种算法的一个主要区别是，除启发值外，A*算法还会考虑在选择下一个节点时到达每一个叶节点的“累计成本”。（对于汽车

的GPS，这一成本是GPS从其内部数据库中获得的行进距离。）因此，A*算法的判定以其对可能的完整路径的成本估计为基础，而不仅仅基于对剩余成本的推测。

基于行为的智能

人工智能早期工作研究的课题涉及直接编写程序来模拟智能。但是，今天许多人认为，人类的智能并非基于复杂程序的执行，而是在经历了世代进化后形成的简单的刺激—反应功能。这种关于“智能”的理论称为基于行为的智能，因为“智能的”刺激—反应功能似乎是一些行为的结果，这些行为导致某些个体在其他个体遇难时得以幸免并能繁衍后代。

基于行为的智能似乎能回答人工智能范畴的若干问题，例如，为什么基于冯·诺依曼体系结构的机器在计算能力上能轻易地胜过人类，却难以展现常识性的判断力。因此，基于行为的智能有希望成为人工智能研究中的一个重要的影响因素。正如文中描述的那样，基于行为的技术已经应用在人工神经网络领域，训练神经元如何按所期望的方式表现；应用在遗传算法领域，为更传统的编程过程提供一个可供选择的方法；应用在机器人领域，通过反应策略来改进机器的性能。

问题与练习

1. 产生式系统在人工智能中有什么重要意义？
2. 画出八数码游戏中，围绕着代表下图状态的节点的那部分状态图：

4	1	3
	2	6
7	5	8

3. 使用广度优先搜索的方法，画出以下图为开始状态解决八数码游戏时控制系统构建的搜索树：

1	2	3
4	8	5
7	6	

4. 用笔、纸以及广度优先方法，构建出以下图为开始状态解决八数码游戏时所产生的搜索树（不必做完），你会碰到什么问题？

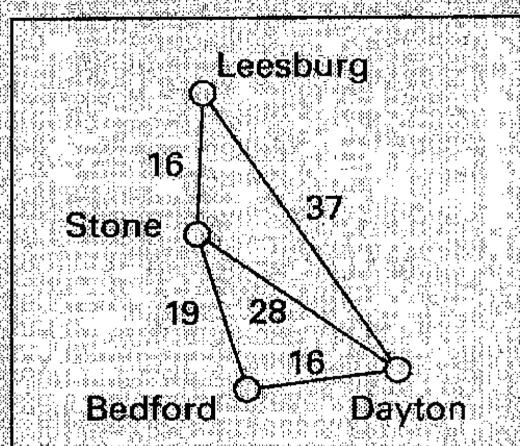
4	3	
2	1	8
7	6	5

5. 登山者为到达顶峰，只考虑当地地形，并总是沿着最陡峭的上坡行进，解决八数码游戏的启发式系统与登山者之间有什么相似之处？
6. 利用本节所讲的启发式方法，采用图11-10所示的最佳控制系统算法，解决下面给出的八数码游戏：

1	2	3
4		8
7	6	5

7. 改进我们计算八数码问题中一个状态的启发值的方法，使图11-10所示的搜索算法不会作出错误的选择，就像在本节中的例子那样。你能否举出一个例子，改进的启发值仍然会导致搜索“误入歧途”？

8. 下面是寻找从Leesburg到Bedford的路线时由最佳适应算法得到的搜索树（图11-10）。这一搜索树中的每一个节点都是地图上的一个城市，其中第一个节点表示Leesburg。当扩展一个节点时，仅添加与当前被扩展的城市直接相连的城市。在每一个节点中记录它到Bedford的直线距离，并将此用作启发值。最终得到的最佳适应算法解决方案是什么？我们得到的是否为最短路线？



从以下城市到Bedford的直线距离：

Dayton	16
Leesburg	34
Stone	19

9. A*算法从两个主要方面修改了最佳适应算法。首先，A*算法记录到每个状态的实际成本。对于地图上的路线，实际的成本便是行进的距离。其次，当选择一个要扩展的节点时，A*算法选择其实际成本与启发值之和最小的节点。根据这两个修改，请绘制问题8的搜索树。在每一个节点中记录行进至这一节点的距离、到达目标城市的启发值，以及它们的和。最终的解决方案是什么？最终的解决方案是最短路线吗？

11.4　其他研究领域

本节，我们来探究人工智能领域一直挑战研究人员的另外两个主题：知识处理和学习。这对人脑来说很简单，但是对机器来说却负担沉重。目前，本质上通过避免直接面对这些问题（或许通过应用聪明的简化方法或限制问题出现的范围）在开发“智能”体方面已经取得了很大的进展。

11.4.1　知识的表达和处理

在关于感知的讨论中，理解图像需要大量的关于图像细节的知识，理解一句话的意思可能要依赖其所处的上下文。这些都是知识仓库起作用的例子，这些称为**真实世界的知识**（real-world knowledge），它们是由人脑维护的。人类以某种方式存储大量的信息并且以极高的效率从这些信息中汲取有用的信息。赋予机器这种能力是人工智能面临的一个重要挑战。

潜在的目标是找到表示和存储知识的途径。这是很复杂的，就像我们已经看到的事实那样，知识以陈述性和过程性这两种形式出现。因此，知识表示不仅是事实的表示，而是包含了一个更广泛的领域。因此，最终能否找到一种用来表示所有形式知识的单一方案是值得怀疑的。

然而，问题不仅仅是表示和存储知识。知识也必须是容易理解的，但获取这种理解是一个挑战。11.2节介绍的语义网通常用来作为知识表示和存储的一种手段，但是，从中提取信息可能是有问题的。例如，句子“Mary hit John”的意思依赖于Mary和John的相对年龄（是2岁和30岁或是相反？）。这种信息可能被存储在图11-3给出的完整的语义网中，但是在上下文分析的过程中，提取这样的信息需要对语义网进行大量的搜索。

处理知识存取还有另外一个问题是，知识的确定不是明确的，而是含蓄的，是与手头的工作相联系的。相对于直接用一个“没有”来回答问题“亚瑟赢了比赛吗？”，我们希望系统可以

这样回答："没有，他因为流感病倒了，没有赢得比赛。"下一节我们将探究联想记忆的概念，这是试图解决有关系的信息问题的一个研究领域。然而，任务不仅仅是找到有关系的信息，我们需要系统能够区分有关系的信息和相关联的信息。例如，"不，他是一月份出生，他妹妹的名字叫丽莎。"这样的回答对于前面问题来说是没有意义的，即使该信息是通过某些相关的方式提交。

开发更好的知识提取系统的另一个方法是向提取过程插入各种各样的推理，结果产生了称为**元推理**（meta-reasoning）的方法，即关于推理的推理。例如，数据库搜索最初是应用**封闭世界假设**（closed-world assumption）。这种假设先假设一个句子为假，除非它能够从可用的信息中明确得出。例如，封闭世界假设允许数据库作出Nicole Smith没有订阅特定的杂志这样的推断，即使数据库中没有包含任何关于Nicole的信息。处理过程就是观察Nicole Smith不在订阅列表中，然后应用封闭世界假设推断Nicole Smith没有订阅。

封闭世界假设表面上看起来是可行的，但是结果证明，元推理发挥的作用是微不足道、不合需要的。例如，假定我们仅有的知识是一句话：

Mickey is a mouse OR Donald is a duck.

只看这个句子，我们不能推断出Mickey实际上是一只老鼠。因此，封闭世界假设强制断定句子

Mickey is a mouse.

为假。以相同的方式，封闭世界假设强制断定句子

Donald is a duck.

为假。这样，尽管两个句子中至少有一个为真，封闭世界假设已经引导我们得出了相矛盾的结论：两个句子都为假。理解这种看似可行的元推理技术的结果是人工智能和数据库这两个领域研究的一个目标，同时也强调了涉及智能系统开发的复杂性。

最后，有一个问题称为**框架问题**（frame problem），用来在变化的环境中使存储的知识保持最新。如果一个智能体打算使用它的知识来决定其行为，那么，该知识必须是当前的。但是，支持智能行为所需知识的数量是庞大的，在变化的环境中维护这些知识是一项繁重的工作。一个复杂的因素是，在一个环境中发生的改变经常会间接地改变信息中的其他细节，而且说明这种间接影响的结果是很困难的。例如，如果一个花瓶被敲碎了，尽管水洒了是打碎花瓶的唯一间接结果，但对于这种状况，你的知识不会再包括水在花瓶中这样的事实。因此，解决框架问题不仅需要以一种有效的方式存储和获取大量信息的能力，而且要求存储系统能够正确地反应间接的推论。

11.4.2 学习

除了表示和处理知识，我们还希望赋予智能体获取知识的能力。我们可以通过编写和安装一个新程序或者显式地向数据库中添加数据来"教"基于计算机的智能体，但是我们更希望智能体能够自己学习。我们希望智能体能够适应环境的变化并执行任务，这些任务并不是通过事先简单地编写程序就能够完成的。一个为做家务而设计的机器人将面对新家具、新设备、新宠物甚至是新主人；一辆能自己驾驶的轿车必须能适应道路边界线的变化；博弈智能体应当能够开发和应用新的策略。

一种把计算机学习的途径进行归类的方法是根据需要人类干涉的程度。学习的第一层是**模仿**（imitation），人类直接演示一个任务的步骤（可能是通过执行一系列的计算机操作或是通过一系列动作将机器人移动），而计算机仅仅是记录这些步骤。这种形式的学习应用在电子表格和

字处理软件等应用程序中已经很多年了，在这些应用软件中，记录频繁发生的指令序列，然后通过一个请求就可以重放。注意，通过模仿学习，智能体承担的任务很少。

学习的下一层是**监督学习**（supervised training）。在监督学习过程中，人对一连串的示例确定正确的反应，然后智能体对这些示例进行归纳，开发出一种适用于新案例的算法。这一连串的示例称为**训练集**（training set）。监督学习的典型应用包括学习识别一个人的笔迹或声音，学习区分垃圾邮件和受欢迎的邮件，以及学习如何通过一组症状判断疾病。

学习的第3层是**强化学习**（reinforcement）。在强化学习过程中，给予智能体一个一般规则，通过反复试验，使它能自己判断任务的成功或失败。当胜负容易界定时，强化学习对于学习如何玩国际象棋或跳棋这样的游戏是很有帮助的。相对于监督学习，当智能体学习改善自身的行为时，强化学习允许智能体自动作出反应。

因为还没有发现指导所有可能的学习行为的通用的学习规则，所以学习一直是一个有挑战性的研究领域。然而，有大量的例子说明人们已经取得了进展。其中一个就是在卡内基梅隆大学开发的ALVINN（基于神经网络的自动驾驶车辆）系统，该系统学习如何驾驶一辆配有一台车载电脑的大篷车，车载电脑使用一台摄像机作为输入。这里所采用的方法就是监督学习。ALVINN从人类驾驶员那里搜集数据并且利用这些数据调整自己的驾驶决策。通过学习，该系统可以预测向哪里驾驶，对照人类驾驶员的数据来检查自己的预测，然后修改自己的参数使其更接近人类的驾驶选择。ALVINN获得了很大的成功，它能够以每小时70英里的速度驾驶大篷车，同时带动了其他方面的研究，已经产生了可以成功在道路上高速驾驶的控制系统。

逻辑程序设计知识

知识表示和存储时很重要的一点就是要采用与存取知识的系统兼容的方式来完成。只有这样逻辑程序设计（参见6.7节）才通常是有利的。在这样的系统中，知识用

Dumbo is an elephant.

和

X is an elephant implies X is gray.

这样的“逻辑”语句来表达。这样的语句能够用符号系统来表达，从而方便地应用推理规则。图11-5所示的演绎推理序列就可以采用直接的方式实现。因而在逻辑程序设计中，知识的表达和存储与知识的提取和应用过程很好地整合在一起。可以说，逻辑程序设计系统为知识的存储和应用提供了完美的衔接。

最后，我们应该认识一个与学习紧密相关的一个现象：发现。两者的区别是，学习是“基于目标的”而发现不是。术语发现含有意料之外的意思，不是现有的可以学习的。我们可以着手去学习一门外语或如何驾驶轿车，但是可能发现那些任务比我们想象得更加困难。探索者可能会发现一个大的湖，而目标仅仅是学习了解那里究竟有什么。

开发具有有效发现能力的智能体要求该智能体能够识别潜在的富有成效的“思考训练”。这里，发现在很大程度上依赖推理的能力以及启发的作用。此外，许多用于发现的应用经常要求智能体能够区别有意义的结果和无意义的结果。例如，一个数据挖掘智能体不应当报告所发现的每一个微不足道的关系。

计算机发现系统中成功的例子包括以哲学家弗朗西斯·培根爵士命名的Bacon，它已经能发现（或许应该说是“重新发现”）电学上的欧姆定律、行星运动的开普勒第三定律，以及动量

守恒定律。系统AUTOCLASS更有说服力，它采用红外光谱数据，已经发现了目前在天文学上未知的新型恒星，这是由计算机完成的一个真实的科学发现。

11.4.3　遗传算法

A*算法（见11.3节）可寻找许多搜索问题的最优解，但有一些问题因其太过复杂无法通过这些技术求解（执行起来超出可用内存，或者无法在合理的时间段中完成）。对于这些问题，有时可以通过一个包含多级试探解的演进过程获得解。这一策略是**遗传算法**（genetic algorithm）的基础。从本质上来说，遗传算法通过随机行为模拟繁殖理论和自然选择的进化过程来求解。

遗传算法首先生成试探解的一个随机池，其中的每个解都只是一种猜测。（对于八数码游戏，试探解可以是方块的一个随机移动序列。）每一个试探解称为一个**染色体**（chromosome），而构成染色体的每个独立个体称为**基因**（gene）——对应八数码游戏中的一次方块移动。

因为每个初始染色体都是一个随机的猜测，所以它不太可能表示手头问题的一个解。因此，遗传算法会生成一个新的染色体池，其中每个染色体是上一个池中两个染色体（父母）的后代（孩子）。这些父母是从池中随机选择出来的，而这个池子也会或然地偏向那些似乎最有可能生成解的染色体，于是这就模拟了适者生存的进化原则。（确定哪些染色体是最佳的父母侯选人，可能是遗传算法过程中最困难的一步。）每个后代都是父母的基因随机组合的产物。此外，最终的后代偶尔可能以某种随机的方式变异（如交掉两次移动）。我们希望，通过反复重复这一过程，将会逐渐演变出越来越好的试探解，直到找出非常好的（甚至是最佳的）试探解。遗憾的是，遗传算法并不保证最终一定能够找到解，但研究证明遗传算法可以有效解决种类繁多的复杂问题。

当被用于程序开发时，使用遗传算法的方法称为**进化规划**（evolutionary programming）。此时，我们的目标是通过模拟进化过程开发程序，而不是直接编写程序。研究人员已经使用函数式程序设计语言将进化规划技巧应用于程序开发过程。运用这个方法，首先要建立一个程序的集合，而这些程序包含各式各样的函数。初始集合中的函数构成了“基因池”，而之后的各代程序将通过“基因池”来构建。接下来，我们可以允许进化过程展开并延续许多代，期望每次通过上一代中的最佳组合生成新的一代，从而让目标问题的解决方案逐步演进。

问题与练习

1. 术语*真实世界知识*是什么意思？它在人工智能领域有何重要意义？
2. 一个关于杂志订阅者信息的数据库通常包含一个每一种杂志订阅者的列表，但是不包含非订阅者的列表。那么，这种数据库如何判断一个人没有订阅一种特定的杂志？
3. 概述框架问题。
4. 给出训练一台计算机的三种途径，哪一种没有涉及直接的人为干预？
5. 进化技术如何区别于更传统的问题解决技术？

11.5　人工神经网络

即使利用人工智能所取得的所有进展，这个领域里的许多问题仍然使得使用传统算法的计算机负担沉重。执行指令序列的中央处理单元的感知和推理能力看来不能与人的大脑相匹敌。由于这个原因，许多研究者的目标转向了利用在事物本质中观察到的现象的方法。其中之一就是上一节介绍的遗传算法，另一种方法就是人工神经网络。

11.5.1　基本特性

人工神经网络提供了模仿活体生物体系统神经元网络的模型。一个生物神经元是单个细胞，具有一些称为树突的输入触角和一个称作轴突的输出触角（如图11-15所示）。经由一个细胞的轴突传递的信号反映了细胞是处于抑制状态还是兴奋状态。这种状态由细胞的树突接收到的信号的组合来决定。这些树突从其他细胞的轴突通过称为突触的小间隙获得信号。研究表明，突触的传导性是由突触的化学成分控制的。也就是说，具体的输入信号将对神经元产生兴奋作用还是抑制作用由突触的化学成分决定。所以可以认为，一个生物神经网络是通过调整神经元间的这些化学连接来学习的。

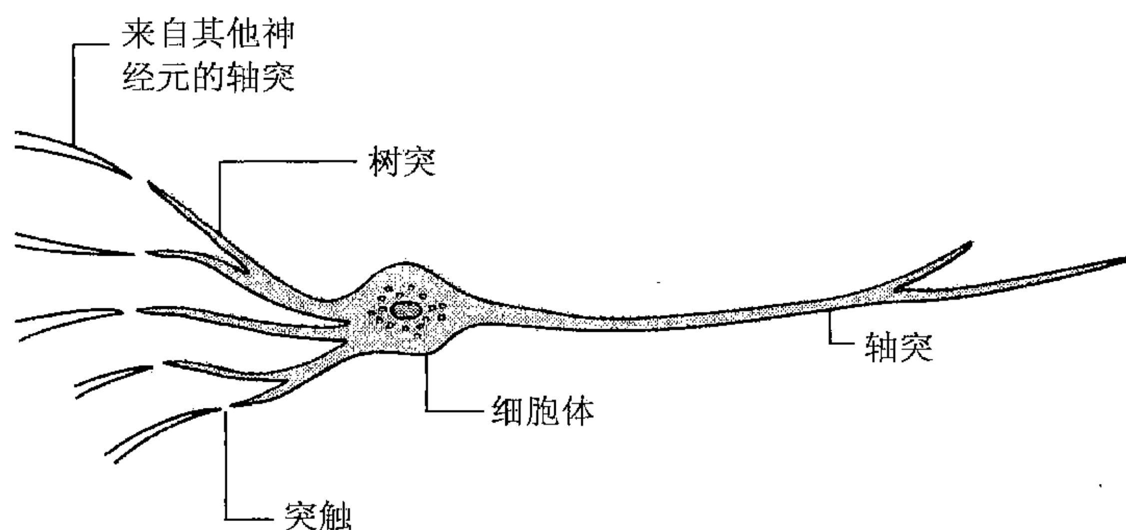


图11-15　活的生物体中的一个神经元

人工神经网络的一个神经元是模仿对生物神经元这种基本了解的一个软件单元。根据其有效输入是否超过了一个给定的值——这个值称为处理单元的**阈值**（threshold value）——产生0或1作为输出。如图11-16所示，这个有效输入是许多实际输入的一个加权和。图中，神经元由椭圆表示，神经元之间的连接由箭头表示。其他神经元（记为v_1、v_2和v_3）的输出用作所描述的神经元的输入。除了这些值，每个连接都与**权**（weight）（记为w_1、w_2和w_3）相关联。神经元把每个输入值与相应的权值相乘，再把这些乘积相加形成有效输入（$v_1w_1+v_2w_2+v_3w_3$）。如果这个和超过该神经元的阈值，那么该神经元就产生一个输出值1（模拟神经元的兴奋状态）；否则就产生一个输出值0（模拟神经元的抑制状态）。

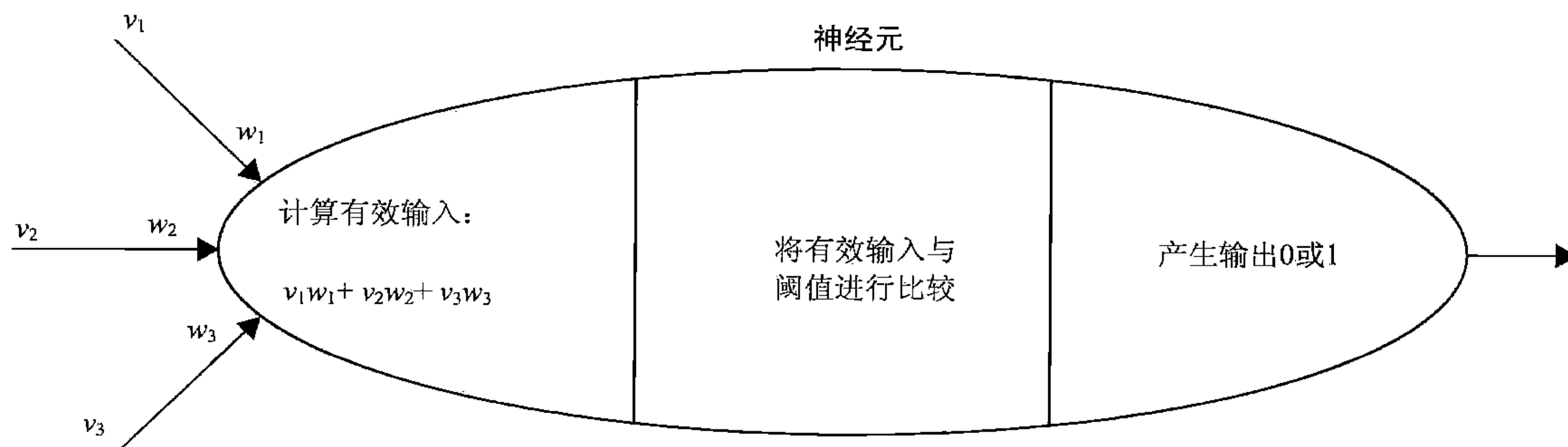


图11-16　一个神经元中的活动

如图11-16所示，我们采用圆形作为表示神经元的约定符号，其中每个输入连接一个神经元，并在圆形内写上与这个输入相关的权值，最后在大圆形中央写上这个单元的阈值。图11-17的例子表示了一个有3个输入且阈值为1.5的神经元。第1个输入的权值为-2，第2个输入的权值为3，

第3个输入的权值为−1。因此，如果神经元接收的输入分别为1、1、0，那么其有效输入为(1)(−2)+(1)(3)+(0)(−1)=1，所以其输出为0。但是，如果神经元接收的输入分别为0、1、1，那么其有效输入为(0)(−2)+(1)(3)+(1)(−1)=2，超出了阈值，所以神经元的输出为1。

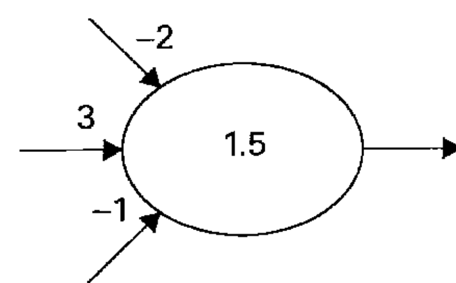


图11-17　一个神经元的表示

权值可以是正值，也可以是负值，说明相应的输入对接收神经元的作用可以是兴奋或是抑制。（若权值为负，接收的输入为1就减少了加权和，故有效输入倾向低于阈值；相反，一个正的权值使相应输入对加权和起增加作用，故增加了加权和超过阈值的机会。）此外，权的实际大小控制了相应输入单元对接收单元起抑制作用或是兴奋作用的程度。因此，通过调节整个人工神经网络中的权值，就能够对网络编程，以预定的方式对不同的输入作出响应。

人工神经网络通常按拓扑节构分层排列。输入神经元位于第一层，输出神经元位于最后一层。其他神经元层（称为隐藏层）可以包含在输入层与输出层之间。一个层中的每个神经元与随后的层中的每个神经元互相连接。图11-18给出了一个简单的神经网络的例子。图11-18a编程为：若两个输入不同，则产生输出1；否则输出0。但如果改变权值如图11-18b所示，那么这个网络无论其两个输入都是1，或者有一个是0，其响应都是1。

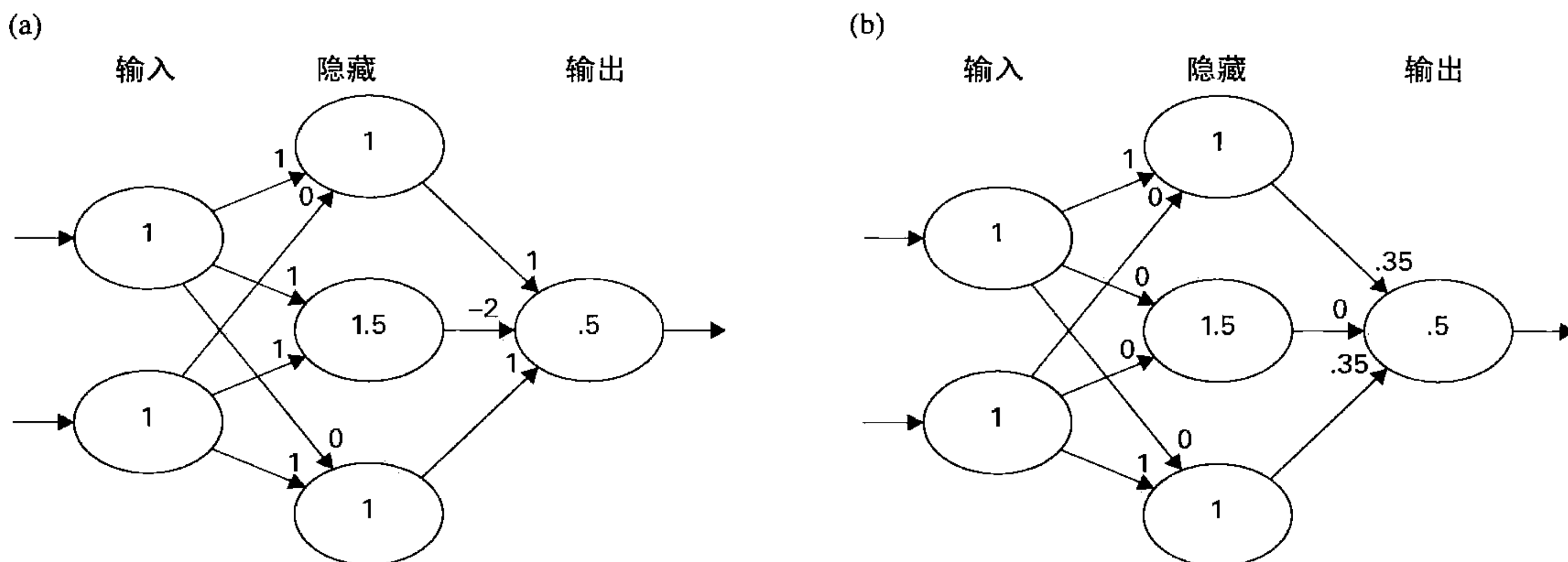


图11-18　有两个不同程序的神经网络

我们应当注意，图11-18所示的神经元网络布局与实际的生物神经网络相比实在是太过简单。一个人的大脑大约包含10^{11}个神经元，每个神经元约有10^{4}个突触。事实上，一个生物神经元的树突多得更像一个纤维网，而不像图11-15中所表示的一个个触角。

11.5.2　训练人工神经网络

人工神经网络的一个重要特征是它不能通过传统意义上的编程而是通过训练实现性能。也就是说，程序员不再决定解决一个特定问题所需的权值，并把这些值“插”入网络中，相反，一个人工神经网络通过监督训练学习获得合适的权值（参见11.4节），该训练是一个反复的过程，从训练装置获得的输入被应用到网络，然后用小的增量调整权值使网络的性能接近期望状态。

值得一提的是，遗传算法技巧已被用于训练人工神经网络。具体来说，要训练一个神经网络，网络的一些权集可以自动随机生成，其中每一个权集用作遗传算法的一个染色体。接下来，网络会被逐步地分得由每一个染色体表示的权值，并通过各种输入进行测试。在这样的测试过

程中，生成错误最少的染色体更有可能被选做下一代染色体的父母。在大量的实验中，这一方法最终都会生成一个成功的权集。

来看这样一个例子，假设成功地训练了一个解决某一问题的人工神经网络，而且这样做可能比尝试通过传统编程方法提供一个解决方案更具有成效。面对这一问题的可能是一个机器人，它通过从摄像头接收的信息理解所处环境。例如，假设机器人必须区分一个房间的墙面（白色）和地面（黑色）。乍一看，你可能觉得这是一个很简单的任务：只需简单分类，将白色像素归为墙面的一部分，将黑色像素归为地面的一部分。然而，当机器人从不同的方位观看或者在房间中走来走去时，不同的光线条件有时会使墙面呈现灰色，而有时又可能让地面呈现灰色。因此，机器人需要学习如何在各种各样的光线条件下区分墙面和地面。

为此，我们可以构建这样一个人工神经网络，其中输入由表明图像中单一像素颜色特征的值和表明整个图像的总亮度的值组成。构建完成之后，我们可以为它提供表示各种光线条件下的墙面部分和地面部分的像素的示例，并以此来训练这一网络。

使用这些方法训练一个人工神经网络的节果如图11-19所示。图中第一列表示原始图像，第二列表示机器人的解释。注意，虽然左上角这一场景中的墙面光线非常暗，但机器人正确地将大多数相关的像素识别为了白色的墙面像素，而下面的一幅图中的地面也被正确识别了。（图像中的球是扩充实验的一部分。）从图中我们还可以看到一点，机器人的图像处理系统并不完美。这一神经网络错误地将一些墙面像素识别成了地面像素（同样，一些地面像素被识别成了墙面像素）。无论应用何种理论，都必须考虑这些的现实的例子。在这种情况下，我们可以通过编程让机器人忽略出现在大量墙面像素中的个别地面像素，并以此来纠正错误（反之亦然）。

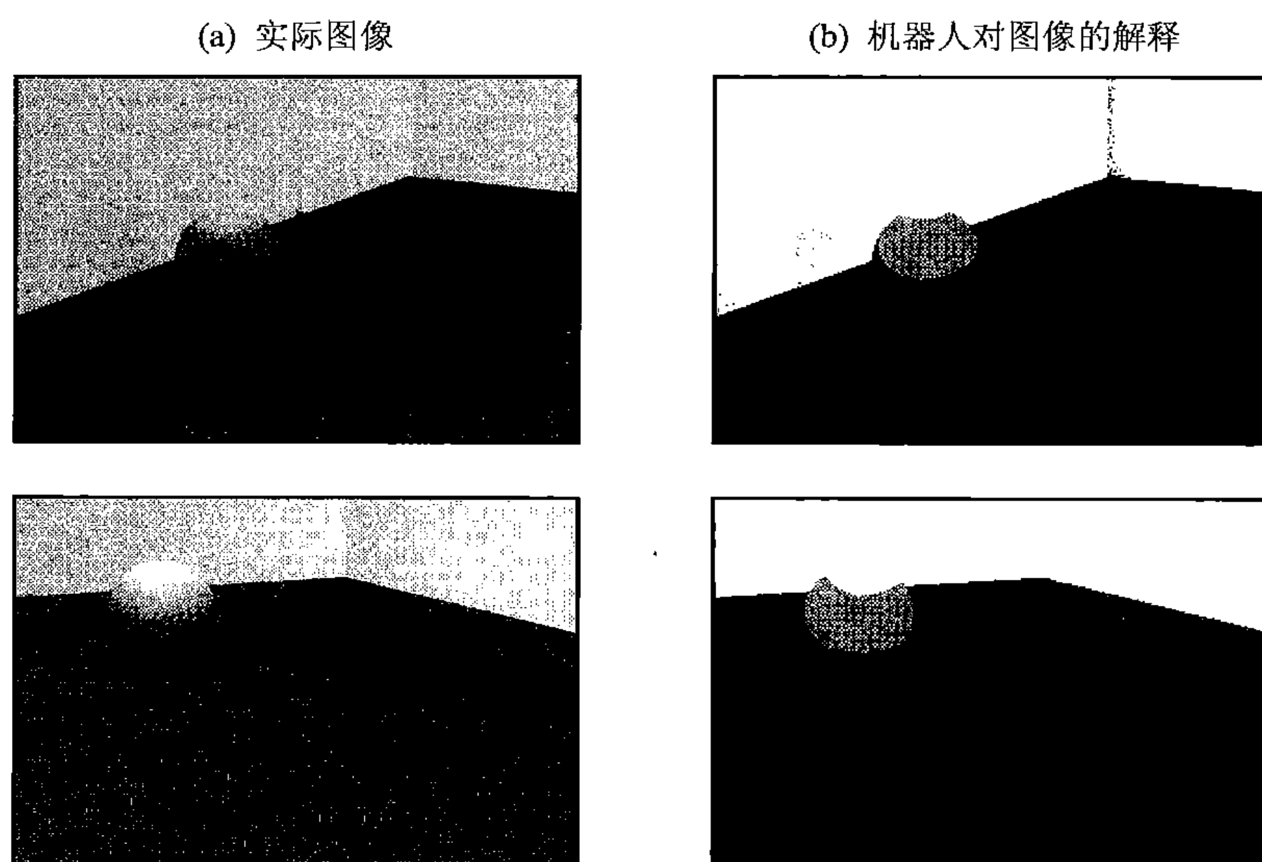


图11-19　使用神经网络分类图像中的像素（图片由www.actapress.com授权）

除了简单的学习问题（如像素分类），人工神经网络已用于学习复杂的智能行为，前一节引用的ALVINN项目就是一例。实际上，ALVINN是一个人工神经网络，如图11-20所示，其构成出奇地简单。ALVINN从30×32阵列的传感器中获得输入，每个传感器负责观察前面道路视频图像的一个特定部分，并且向隐蔽层的4个神经元的每一个报告其发现。（从而，这4个神经元都有960个输入。）每一个神经元的输出与30个输出神经元的每一个相连接，其输出指明了驾驶的方向。这30个神经元形成一行，一端处于兴奋的神经元表示向左急转弯，而另一端处于兴奋的神经元则表示向右急转弯。

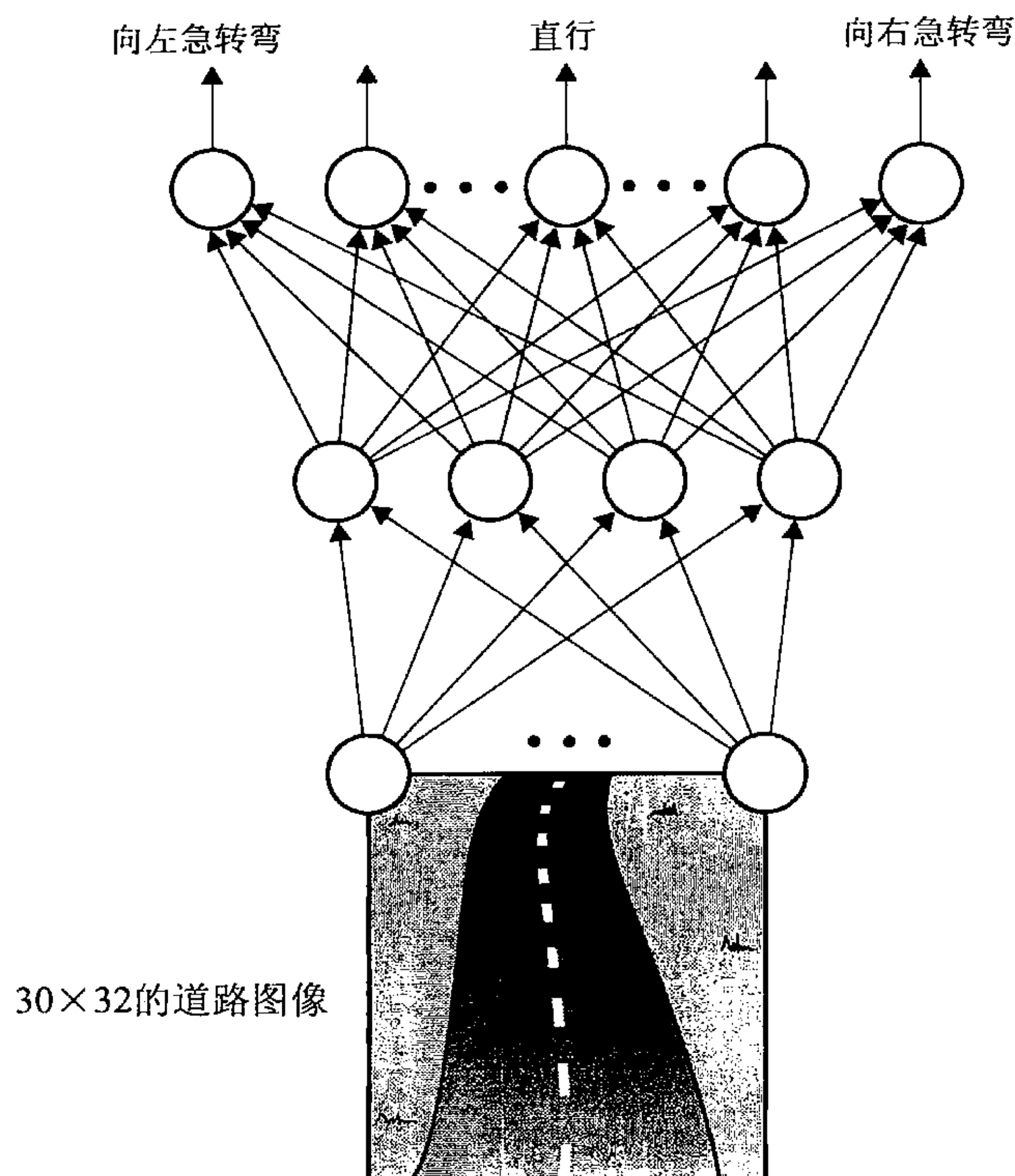


图11-20 ALVINN的结构

ALVINN的训练是通过“观察”一个人的驾驶进行的。当要做出自己的驾驶决定时，它把自己的决定与人的决定进行比较，并且稍微修改其权值使其更接近人的决定。然而，存在一个有意思问题，尽管ALVINN通过它的简单技术学习驾驶，但是它没有学习如何从错误中恢复过来。因此，从人那里收集的数据也要人为加强恢复状态。(最初考虑的恢复训练的一种方法是使人令交通工具偏离方向，以便ALVINN可以通过观察人来学习如何恢复。但在人进行初始的偏离过程时应该让ALVINN关闭，否则，ALVINN除了恢复也会学会偏离——显然，这并不是一个受欢迎的方法。)

11.5.3 联想记忆

人脑具有惊人的能力，能够从当前关心的情景中提取与之关联的信息来。当闻到特定气味时，我们很容易勾起对儿时的回忆；朋友的声音会唤起友人的身影和一段美好时光的回忆；特定的音乐可能会产生对某个假日的怀念。这些就是**联想记忆**（associative memory）——提取与手头信息相关的信息。

构建具有这种联想记忆能力的机器是许多年来研究的一个目标。途径之一是应用人工神经网络技术。例如，考虑一个由许多神经元组成的网络，这些神经元相互连接形成一个没有输入和输出的网。有些设计中，一个神经元的输出连到其他神经元，作为这些神经元的输入，这种设计称为霍普菲尔德网络（Hopfield network）。在有些设计中，一个神经元的输出可能只连到与其直接相邻的神经元。在这样的系统中，处于兴奋状态的神经元将会使其他神经元进入兴奋状态，而处于抑制状态的神经元将会使其他神经元进入抑制状态。因此，整个系统可能会处于不断发生变化的状态中，也可能会找到办法实现稳定格局（其中，处于兴奋状态下的神经元会保持兴奋状态，而处于抑制状态下的神经元会保持抑制状态）。如果以一个接近某一稳定格局的不

稳定格局启动网络，我们可以期望它进入稳定格局。从某种意义上说，当给定一个稳定格局的一部分，网络有可能完成这一格局。

现在假设我们用1表示一个活跃状态，0表示抑制状态，这样任何时刻的整个网络的状态都能被想象成0和1的布局。然后，如果把网络设置为一个接近稳定模式的位模式，我们就可以期望网络转换到稳定模式。换言之，网络可能找到接近赋予它的模式的稳定位模式。所以，如果一些位用来编码成“气味”，另一些位用来编码成“儿时回忆”，那么，根据某个稳定布局初设的“气味”，能够使其余的位找到关联的“儿时回忆”。

现在我们考虑图11-21所示的人工神经网络。遵照用于描述人工神经网络的约定，图中每个圆圈代表一个处理神经元，其阈值记于圆中。连接圆圈的线（而不是箭头）代表相应神经元间的双向连接。也就是说，一条连接两个神经元的线表示每个神经元的输出连到另一个神经元作为输入。因此，中央神经元的输出连到其周边每个神经元作为输入，而周边每个神经元的输出也都连到中央的神经元作为输入。两个相连的神经元相互的输出都有相同的权值。这个共同的权值记在连接线旁边。于是，图中顶部那个神经元从中央神经元接收的输入伴有权值-1，从其两个周边邻居接收到的输入伴有权值1。类似地，中央神经元从周边各神经元接收的输入伴有权值-1。

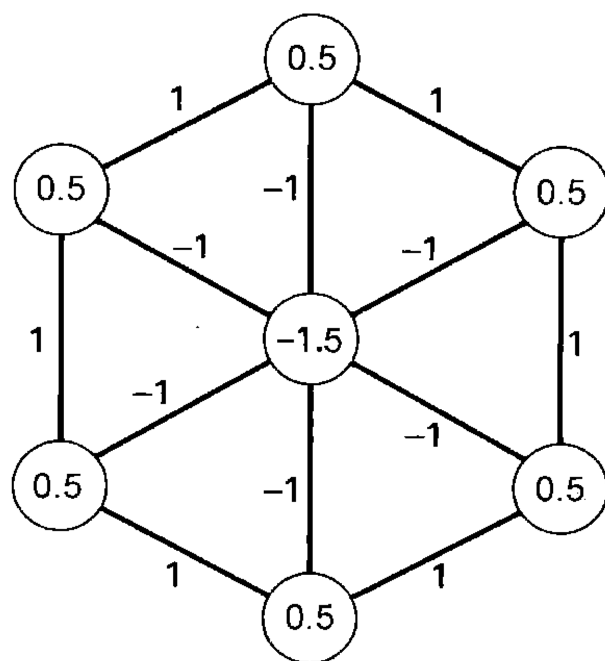


图11-21　实现联想记忆的一个人工神经网络

网络以独立的步骤运转，每一步，所有的神经元都以同步方式对其输入作出响应。为了从网络的当前布局确定其下一步布局，我们要确定整个网络中每一个神经元的有效输入，再让所有的神经元同时响应其输入。结果，整个网络遵循一个协调的顺序运作：计算有效输入，响应输入，计算有效输入，响应输入，依次类推。

如果网络初始化为最右边两个神经元为抑制状态，其他神经元为兴奋状态（如图11-22a所示），我们来考虑会发生哪些事件。最左边两个神经元的有效输入为1，所以保持兴奋；但它们周边邻居的有效输入为0，所以会变成抑制状态。类似地，中央神经元的有效输入为-4，所以变为抑制状态。于是，整个网络转变成图11-22b所示的布局，只有最左边两个神元处于兴奋状态。因为中央神经元现在为抑制状态，所以最左边两个神经元的兴奋状态将导致顶部和底部两个神经元再次变成兴奋状态。同时，因为有效输入为-2，中央神经元继续保持抑制状态。于是网络转变成图11-22c所示的布局，然后它又导致了图11-22d所示的布局。（如果网络初始化为只有上面4个神经元为兴奋状态，那么会出现一种闪烁现象。顶部神经元保持兴奋，而其2个周边邻居及中央神经元会在兴奋与抑制两种状态间不断切换。）

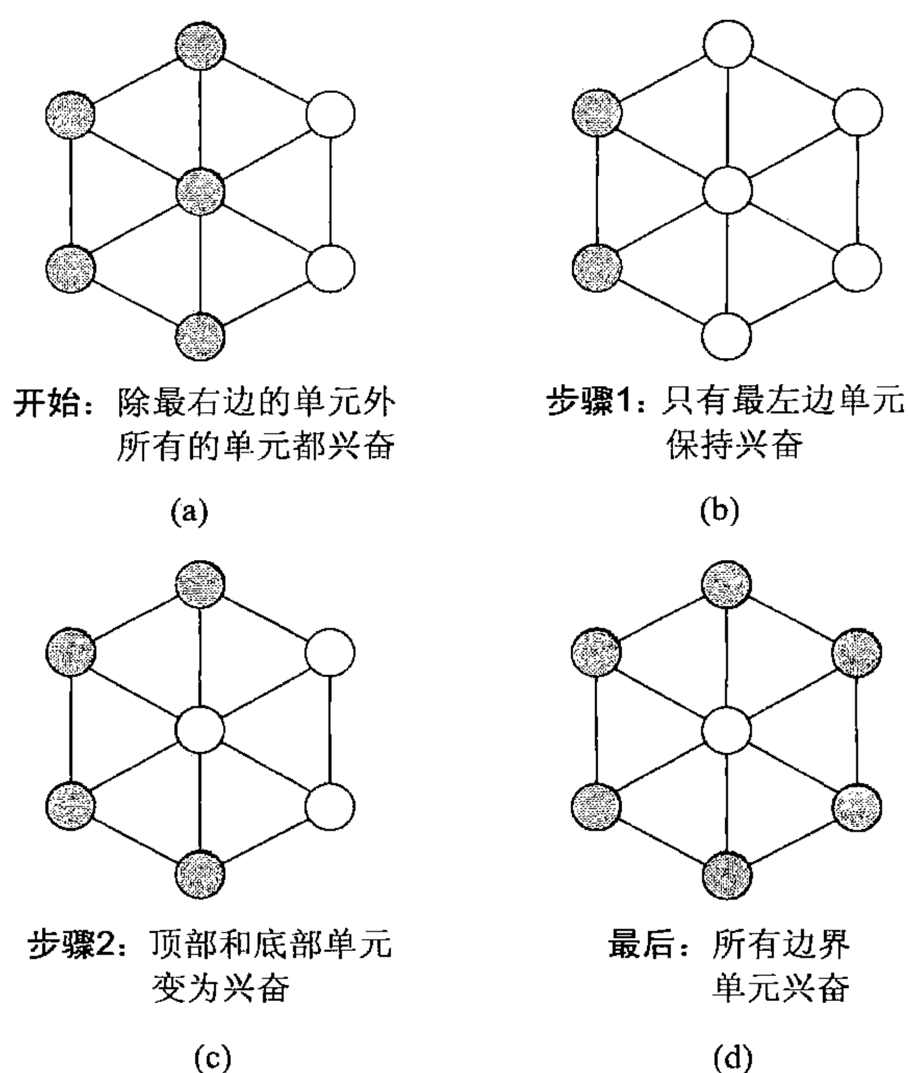
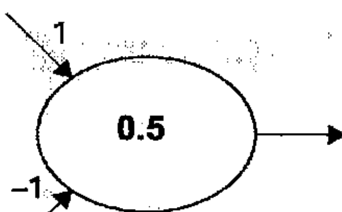


图11-22 导向稳定布局的步骤

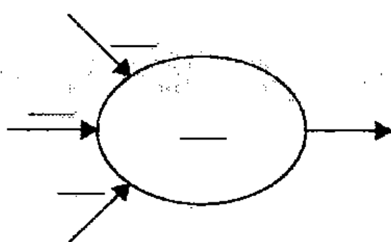
最后，我们观察到这个网络有两种稳定布局：一种是中央神经元处于兴奋状态，而其他神经元处于抑制状态；另一种是中央神经元处于抑制状态，而其他神经元处于兴奋状态。如果网络初始化為中央神经元兴奋而其他处于兴奋状态的神经元不超过两个，那么网络会走向前一种稳定布局。如果网络初始化为至少4个相邻周边神经元处于兴奋状态，那么网络会走向后一种稳定布局。所以可以说，这种网络，如果初始模式为中央神经元及3个以下周边神经元处于兴奋状态，就与前一种稳定布局相关联；如果初始模式为4个或4个以上周边神经元处于兴奋状态，那么就与后一种稳定布局相关联。简单来说，这个网络表示基本的联想记忆。

问题与练习

1. 对于下面的神经元，当两个输入都是1时，输出是多少？如果输入模式是0,0、0,1以及1,0呢？



2. 调整下列神经元的权和阈值，使得当且仅当至少有两个输入为1时输出为1。



3. 举例说明在训练一个人工神经网络时可能会发生的一个问题。
4. 在图11-22中，如果初始化为所有神经元处于抑制状态，那么网络会走向哪个稳定布局？

11.6 机器人学

机器人学（robotics）是研究具有智能行为的物理上的自主智能体的一门学科。对于所有的智能体，机器人在所处的环境中必须能够感知、推理和发生作用。因此，机器人学涵盖了人工智能的所有研究范围，并借鉴了很多机械和电子工程方面的知识。

机器人需要用机械装置来回移动和操作目标物体来与外界交互。在早期的机器人学中，该领域与操作器的发展联系紧密，这些操作器通常是带有肘、腕及手或工具的机械臂。研究不仅涉及这样的装置如何操作，而且涉及如何维护和应用有关它们的位置和方向的知识。(你闭上眼睛也能够用手摸到自己的鼻子，因为你的大脑保存有鼻子和手指在什么地方的记录。）随着时间的推移，机械臂已经能够更灵巧地定位，使用基于力反馈的触觉，机械臂能够成功地握住鸡蛋和纸杯。

最近，快速、轻便计算机的发展促进了移动机器人方面更重大的研究。这种灵活性导致了大量富有创意的设计。在机器人移动能力方面，研究人员已经开发出可以像鱼一样游动、像蜻蜓一样飞翔、像蝗虫一样跳跃、像蛇一样蜿蜒爬行的机器人。

因为带有轮子的机器人相对容易设计和建造，所以非常受欢迎，但是它会受地形的限制。结合使用轮子和导轨，克服这种限制，使机器人能够爬楼梯或翻越岩石是当前的研究目标。例如，NASA（美国国家航空航天局）的“火星探路者”就是使用特殊设计的轮子在火星的岩石层上行走。

有腿的机器人其可移动性大大提高，但是相当复杂。例如，设计能像人一样行走的两条腿机器人必须持续地监视和调整其姿态，否则它会跌倒。但是，这种困难能够克服，例如本田公司开发的两条腿的具有人类特征的机器人Asimo，能够上楼梯，甚至能够跑。

尽管在操作器和移动能力方面取得了巨大的进步，但是大多数机器人仍然不是非常自主的。通常工业机械臂是为每个任务通过严格编程设计出来的，工作时不用传感器，它假设零件将会按照指定的位置精确地传送给它们。其他的移动机器人（如NASA的“火星探路者”和军用无人机）其智能依靠人的操作来实现。

克服这种对人的依赖是当前研究的一个主要目标。问题涉及一个自主的机器人需要知道关于其所处环境的哪些知识，以及需要预先计划其行为到什么程度。一种方法是建造能维持所处环境详细记录的机器人，该记录还包含目标物体的一个详细目录以及它们的方位，通过这些信息制定行动的详细计划。这个方向的研究很大程度上依靠知识表示和知识存储方面取得的进展以及推理和规划技术的改进。

另一个可选择的方法是开发反应型机器人，该方法不用保持复杂的记录，也不用耗费精力去构建详细行动计划，只要应用简单的与外界交互的规则时时刻刻指导它们的行为即可。反应型机器人技术的支持者认为：当计划一个长途汽车旅行时，人们不会预先制定全面而详细的计划，相反，他们仅是选择主要路线，而对于像到哪吃饭、走哪些出口，以及如何绕道行驶等细节则到时候再考虑。同样，一个反应型机器人若要通过一条拥挤的走廊，或从一栋大楼走到另一栋大楼，也不会预先制定非常详细的计划，但是当碰到障碍时，它会应用简单的规则避开。这是历史上最畅销的机器人——iRobot Roomba真空吸尘器所采用的方法，真空吸尘器以反应模式在地面上来回移动，而不会为记住家具的详细信息和其他障碍而费心。毕竟，这次碰到的家庭宠物下次不可能还呆在同一个地方。

当然，单一的方法并不是对于所有情况都是最好的。真正的自主机器人最有可能使用多层标准的推理和计划，应用高级技术设定和达到主要目标，应用低级反应系统完成次要目标。这种多层次推理的例子可在RoboCup比赛（一个机器人足球队的国际性比赛）中发现，人们力图

在2050年开发出能够对抗世界级人类足球队的机器人足球队。这里，重点不仅是建造能够“踢”球的移动计算机，而是设计一个能够相互协作达到共同目标的机器人足球队。这些机器人不仅要移动和对自己的行为做出推断，而且还要对队友和对手的行为作出推断。

机器人学研究领域的另一个例子是进化机器人学，该领域把进化理论应用于开发低级反应规则和高级推理所对应的方案。这里我们发现，适者生存理论用到了设备的开发上，经过若干代的学习，这些设备已经能够自己获得平衡或移动的方法。这个领域的许多研究各有侧重，其不同之处在于机器人的内部控制系统（很大程度上是软件）及其形体的物理结构。例如，把一个能游泳的蝌蚪机器人的控制系统换在一个有腿的类似机器人身上，然后在控制系统中应用进化技术，就得到一个能爬行的机器人。还有一些例子中，进化技术已经被应用在机器人的物理形体上，让传感器发现执行特定任务的最佳位置。更具挑战性的研究正在寻求软件控制系统与形态结构同时发展的途径。

机器人学的研究成果很多都令人难忘，这方面的例子举不胜举，当前的机器人与科幻电影和小说中的超能机器人相差甚远，但是在执行特定任务上已经取得了重大成功。机器人已经能够驾驶交通工具，模仿宠物狗的行为举止，或者为武器导航。然而，享受这些成功的同时，我们应该注意，对人造宠物狗的钟情以及智能武器的可怕威力带来了社会问题和道德问题，这些都向社会发出了挑战。我们的未来是我们自己造就的。

机器人创造史

(a) 2010年4月15日，在德国东部的马格德堡，一个机器人正在德国RoboCup 2010公开赛中踢球（© Jens Schlueter/SFP/gettyimages/Newscom）。(b) Tartan Racing团队的赛车“Boss”——DARPA举办的无人驾驶汽车城市挑战赛Urban Challenge的大奖得主。(c) NASA的一台探测器——探测火星表面地质情况的一台机器人。



问题与练习

1. 对于机器人行为的反应方法在何种方式上有别于更传统的“基于计划”的行为？
2. 当前机器人学领域研究的主题有哪些？
3. 进化理论用在机器人开发的哪两个层次？

11.7 后果的思考

毫无疑问，人工智能领域取得的进步可能会造福人类，人们很容易热衷于这些潜在的好处。然而，这也为将来留下了后患，它的破坏性后果与其带来的好处同样巨大。这种差异常常仅在于一个人的观点或一个人的社会地位的不同——彼之所得，此之所失。所以花一点时间从另外一个角度观察正在进步的技术对于我们来说是比较恰当的。

有些人把技术的进步看成是给与人类的一份厚礼——将人类从枯燥的、普通的任务中解放出来，为更愉悦的生活方式打开大门。但对于同一个现象，另一些人则把它看作是剥夺公民就业机会、把财富引向权势人物的祸根。其实，这正是印度忠诚的人道主义者圣雄甘地所预言的。甘地再三地辩称，如果用农夫家庭手纺车来代替大型纺织工厂，那么印度人的生活将会变得更好。他断言，通过这个途径，可以用一个分散的大宗生产系统取代只能雇用少数人的集中式大宗生产，这将有利于平民大众。

历史上有很多因财富和权力分配不均而引起的革命。如果今天正在进步的技术使得这种悬殊更为牢固，那将产生灾难性的后果。

但是，建造越来越智能的机器的后果，比对付不同社会族群间的权力斗争的后果更加微妙，也更加根本。这些问题触及了人类自身形象的核心。19世纪，达尔文的进化论及人类可能由更低等的生命形式进化而来的想法震惊了整个社会。那么，面对机器的智力向人类智力挑战的冲击，社会将如何反应呢？

过去，技术发展缓慢，有时间让我们重新调整智能的概念，维护人类自身形象。19世纪，我们的老祖宗会认为当时的机械装置具有超自然的能力，而今天我们决不会认为这些机械有什么智能。但是，如果机器真的挑战了人类的智能，或者更有可能的是机器能力的进步远远超过了我们的适应能力，那么人类将如何应对呢？

考虑一下20世纪中叶社会对当时IQ测试的反响，也许可以从中得到一些线索，看看人类面对挑战我们智能的机器时的可能反应。这些测试可用来确定孩童的智力水平。美国的孩童常常依据他们在测试中的表现来分类，并据此制定相应的教育计划。随之，受教育的机会向那些在测试中表现良好的孩子开放，而那些测试表现差的孩子只能安排他们去参加补习。简而言之，当给出一种标准来衡量个体的智能时，社会会倾向于漠视被认为是低于这个标准的那些人的能力。那么，如果机器的“智能”能力已经变得可与人类相匹敌，甚至只是看上去可以相匹敌，社会将怎样应对这种局面呢？抑或社会也漠视那些能力看上去不如机器的人？如果这样，对于社会的这些成员来说，后果是什么？难道一个人的尊严会受他与机器的比较结果的影响吗？

我们已经看到，在一些特定场合，人类的智力正面临机器的挑战。机器现在有能力打败棋王，计算机化的专家系统能够给出治疗意见，管理证券投资的简单程序常常比投资专家做得更好。这样的一些系统怎样影响所涉及人员的自身形象？随着在越来越多的领域里机器的表现优于人类，个人的自尊心会受到怎样的影响？

由于人是生物，而机器不是，所以许多人认为机器拥有的智能与人类的智能有着本质的区别。因此他们认为机器永远不会再生出人类的决策过程。机器也许会得到与人同样的结论，但是得到这些结论所依赖的基础与人类并不相同。那么在怎样的程度上存在不同类型的智能？对于社会来说，如果遵循非人类智能提出的道路运作，是否合乎道德？

Joseph Weizenbaum在他的*Computer Power and Human Reason*一书中坚决反对不加抑制地应用人工智能，他这样写道：

计算机能够作出司法判决，计算机能够作出精神病判定。它们能够以比最有耐心的人更加老练的方式投掷硬币。关键在于不应当给它们这些任务。在某些场合，它们甚至能够得到“正确的”决策——但是其依赖的基础总是而且一定不是人类所乐于接受的。

已经有很多有关“计算机与人脑”的争辩。我在这里的结论是，问题的实质不在于技术，甚至也不在于数学，而在于伦理道德。设定计算机去做什么，不能用“能够不能够”这样的问题，计算机适用性的限制最终只能根据“应当”来表达。最基本的认识应该是：因为我们现在还没有办法让计算机有智慧，所以我们现在就不应当让计算机去做有智慧的工作。

也许你会认为本节所述的许多内容近乎科幻小说，而不是计算机科学。就在不久前，许多人因抱有同样的“这永远不会发生”的态度，而拒绝考虑“如果计算机操纵了社会，会发生什么”。但从许多方面来看，这一天现在已经来临。如果一个计算机化的数据库错报了你有不良的信用度、有犯罪记录或是银行账户透支，那么，是计算机的报告会奏效还是你自己的清白申诉会奏效？如果一个不正常的导航系统错误指示了大雾笼罩的跑道位置，那么飞机将降落在何处？如果一个机器用来预测公众对不同政治决策的反应，那么一个政治家应采取何种决策？你遇到过多少次因为“计算机坏了”所以服务员无法为你服务的情形？那么，究竟谁掌管着这个社会？我们还没有准备让社会屈从于机器吗？

问题与练习

1. 如果把过去100年来发明的所有机器都去掉，那么今天的人还有多少能幸存？如果是过去50年呢？20年呢？幸存者会在何处？
2. 你的生活在多大程度上被机器所控制？谁又控制着这些影响你生活的机器？
3. 你从哪里获得那些作为你的日常决策基础的信息？对于你的重大决策呢？对这些信息的准确度你有多少把握？为什么？

复习题

（带*的题目涉及选读小节的内容。）

1. 正如11.2节说明的那样，人类会用一个问题来表达某个目的，而不是提问。另一个例子，“你知道你的轮胎漏气了吗”，这也是用来提醒而不是问。给出一些问题的例子，目的是用来表达安慰、警告或责备。
2. 把一个苏打水剂量器当作一个智能体来进行如下分析：它的传感器是什么？它的效应器是什么？它可以展现什么级别的反应（本能反应、基于知识的反应或基于目标的反应）？
3. 确定下列每一个反应，是本能反应、基于知识的反应还是基于目标的反应。证明你的回答。
   a. 一个计算机程序把文本从德文翻译成英文。
   b. 当室内温度低于当前设定值时，自动调温器打开暖气。
   c. 一位飞行员驾驶飞机完全地在跑道上着陆。
4. 如果一个研究人员使用计算机模型来研究人脑的记忆能力，那么为机器所开发的程序必定要达到机器的最佳存储能力吗？请解释。
5. 举出几个陈述性知识的例子。举出几个过程性知识的例子。

*6. 在面向对象程序设计的环境中，一个对象的哪些部分用来存储陈述性知识？哪些部分用来存放过程性知识？

7. 下列活动中，你认为哪些是面向性能的？哪些是面向模拟的？
   a. 一个自动往返系统的设计（通常用在机场航站楼之间）。
   b. 用于预测台风路径的模型的设计。

c. 一个用于提取和维护万维网上存储的文档目录的Web搜索数据库的设计。

d. 一个用于理论测试的国家经济模型的设计。

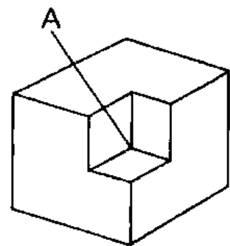
e. 一个用于监视病人生命体征的程序的设计。

8. 当今，某些打给企业的电话由自动应答系统来处理，该系统利用语音识别与打电话的人进行通话。这些系统通过了图灵测试吗？请解释你的答案。

9. 确定能用来区分符号F、E、L和T的一组几何特征。

*10. 请描述通过与模板比较鉴别特性的技术与第1章介绍的利用纠错码鉴定特性的技术之间的相似之处。

11. 根据下面线绘图中标记A的那个角是凸起还是凹下，说明这个图的两种解读。



12. 比较下列两个句子中介词短语的作用(仅有一个词不同)。如何对一台机器编程让它做这样的区分？

The pigpen was built by the barn.
The pigpen was built by the farmer.

13. 下面两个句子的语法分析的结果有什么不同？语义分析的结果有什么不同？

An awesome sunset was seen by Andrea.
Andrea saw an awesome sunset.

14. 下面两个句子的语法分析的结果有什么不同？语义分析的结果有什么不同？

If X<10 then subtract 1 from X else add 1 from X.
If X>10 then add 1 to X else subtract 1 from X.

15. 正文中，与形式程序设计语言相比，简要讨论了理解自然语言的问题，作为讨论自然语言所涉及的复杂性方面的例子，给出问题“Do you know what time it is?”有不同含义的情形。

16. 一个句子上下文的改变能够改变这个句子的含义以及意思。在图11-3的上下文中，如果两人都出生于21世纪晚期，那么句子“Mary hit John.”的含义怎样改变？如果一个出生在20世纪80年代，而另一个出生在21世纪晚期，那么含义如何改变？

17. 画一个语义网，把下列段落的意思表示出来。

Donna threw the ball to Jack, who hit it into center field. The center fielder tried to catch it, but it bounced off the wall instead.

18. 有时候回答一个问题的能力依赖于知识水平，这与对事实本身的依赖程度相同。例如，假定数据库A和B都包含一个完整的雇员名单，该名单与公司健康保险程序相关联。但是只有数据库A知道名单是完整的。那么关于一个不在名单里的员工，数据库A能够推断出的什么信息是数据库B推断不出来的？

19. 举出一个封闭世界假设导致矛盾的例子。

20. 举出两个共用一个封闭世界假设的例子。

21. 在产生式系统中，状态图和搜索树有什么区别？

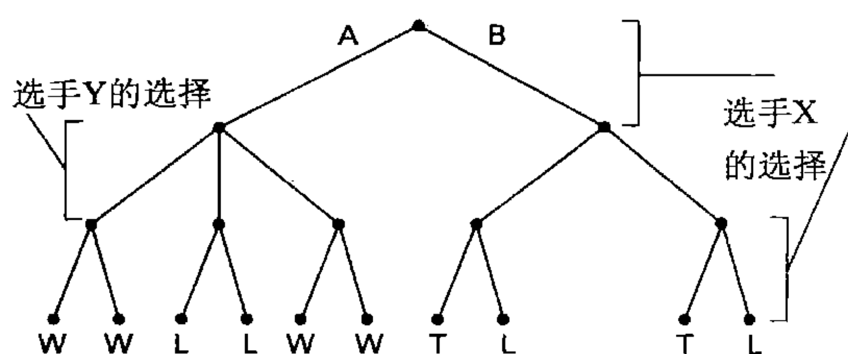
22. 依照一个产生式系统分析解决魔方问题的任务。(什么是状态？什么是产生式？)

23. a. 假定搜索树是一个二叉树，达到目标需要8个产生式。如果该树是以广度优先的方式构建的，那么当达到目标状态时，树中最多的节点数是多少？

b. 解释通过同时构建两个搜索如何能够减少搜索过程中考虑的全部节点数目——一个搜索从初始状态开始，同时另一个搜索从目标状态逆向进行直到这两个搜索会合。(假设记录在逆向搜索过程中发现的状态的搜索树也是一个二叉树，并且两个搜索以相同速度进行搜索。)

24. 正文中我们提到，产生式系统通常被用来作为从已知事实中得出结论的一种技术。系统的状态是推理过程的每一个阶段认为是真的事实，产生式对于操纵已知事实来说是逻辑规则。标识几个逻辑规则，支持从事实“John is a basketball player”，“Basketball player are not short”，以及“John is either short or tall”中能够得出结论“John is tall”。

25. 下面的树表示一个竞赛游戏中可能的移动，选手X当前可在移动A和移动B中选择其一。选手X移动后，选手Y跟着选择移动，然后由选手X来移动最后一步。树的叶子节点标记为W、L或T，分别代表选手X最后是赢、输还是平局。选手X应选择移动A还是移动B？为什么？在一个竞赛性的游戏中选取一个“产生式”和八数码游戏等单人游戏中的选取有什么不同？



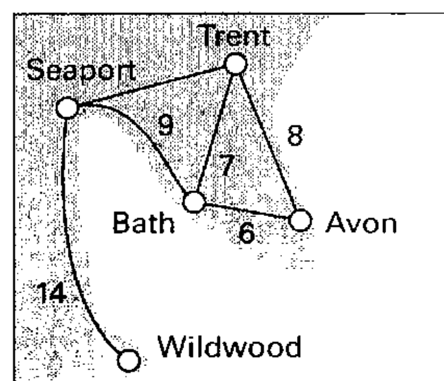
26. 按照产生式系统分析跳棋游戏，并描述一个用来在两个状态中确定一个更接近目标的状态的启发。这种情况中的控制系统与一个八数码游戏等单人游戏中的控制系统有什么区别？
27. 把代数定律看作产生式，代数表达式简化的问题就能在产生式系统的上下文中解决。确定一组代数产生式，使等式$3/(2x-1)=6/(3x+1)$简化为$x=3$。当进行这种代数简化时，经验法则（即启发法则）是什么？
28. 不用任何启发信息的帮助，画出利用广度优先搜索方法解决如下初始状态的八数码游戏生成的搜索树。

	1	3
4	2	5
7	8	6

29. 利用图11-10的最佳算法解决第28题的八数码游戏，用未到达正确位置的方块的数目作为启发信息，画出搜索树。
30. 利用图11-10的最佳算法解决如下初始状态的八数码游戏，假设使用与11.3节中一样的启发信息，画出搜索树。

1	2	3
5	7	6
4		8

31. 当解决八数码游戏时，为什么用未到达正确位置的方块的数目作为启发信息不如11.3节用的那种好？
32. 执行二叉树搜索（参见5.5节）时决定考虑哪一半列表的技术，和执行一个启发搜索时决定要执行哪个分支的技术，二者有什么不同？
33. 注意，如果一个产生式系统的状态图中有一个状态的启发值与其他状态相比极其低，并且如果从这个状态到自身有一个产生式，那么图11-10的算法会陷入一个循环，一遍又一遍地考虑这个状态。说明如果执行该系统中任何产生式的代价至少为1，那么把启发值加上沿遍历的路径到达该状态的代价，通过这样计算预测的代价，就可以避免这种无限循环。
34. 在一幅大的交通图上寻找两个城市间的道路，你会用怎样的启发。
35. 请看在寻找从Trent到Wildwood的路线时，根据图11-10中的最佳适应算法得出的四层搜索树。这一搜索树中的每个节点表示地图上的一个城市。开始节点为Trent。当扩展一个节点时，仅添加与被扩展的城市直接相连的城市。在每个节点中记录到Wildwood的直线距离并将其用作启发值。在其处理过程中，最佳适应算法有什么不足之处吗？如果有，应该怎样纠正？



此处距Wildwood的直线距离：

Avon	10
Bath	8
Trent	15
Seaport	13

36. A*算法从两个主要方面修改了最佳适应算法。首先，A*算法记录到每个状态的实际成本。对于地图上的路线，实际的成本便是行进的距离。其次，当选择一个要扩展的节点时，A*算法选择其实际成本与启发值之和最小的节点。根据这两个修改，请绘制问题35的搜索树。在每一个节点中记录行进至这一节点的距离、到达目标城市的启发值，以及它们的和。从Dearborn到Wildwood的路线是什么呢？
37. 列出可用于产生式系统的启发所具有的两个特性。
38. 假定有两个桶，一个容量是3升，一个容量是5升。任何时候你都可以把水从一个桶倒入另一个桶，把一个桶倒空，或把一个桶倒满。问题是要将正好4升的水注入5升的那个桶。说明这个问题如何可以设计成一个产生式系统。
39. 假设你的任务是监督两辆卡车装货，每辆车最多可载14吨货。货物装在不同的筐里，总重28吨，但每一筐的重量不一样，都标在各自的筐边上。为了在两辆车上分装这些货物，你会采用什么样的启发式？
40. 下列哪些是元推理的例子？

a. 他还没走多长时间所以没走远。

b. 因为我经常做出错误的决定，而所作的最后两个决定是正确的，那么我将逆转下一个决定。

c. 我有些疲倦了，所以我可能不会清晰地思考。

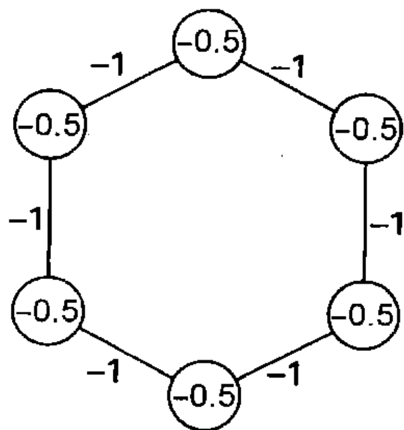
d. 我有些疲倦了，所以我想我将要打个盹。

41. 描述人类解决框架问题的能力如何帮助人类找到丢失的项目。

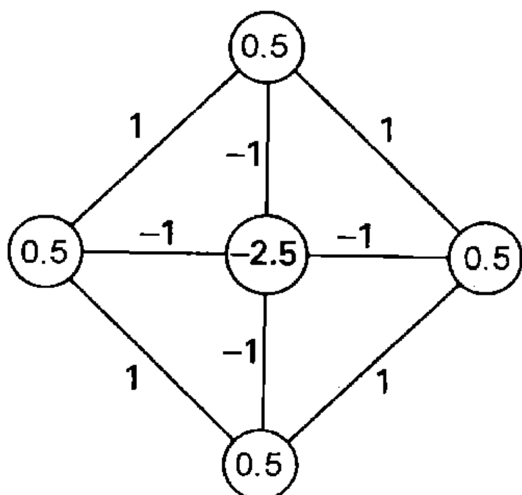
42. a. 模仿学习与监督学习在何种意义上相似？

b. 模仿学习与监督学习在何种意义上不同？

43. 下图表示一个用于11.5节讨论的联想记忆的一个人工神经网络。如果模式中只有两个神经元处于兴奋状态，而这两个神经元被一个神经元分开，那么它与什么模式相关联？如果网络初始时所有单元都处于抑制状态，会发生什么情况？



44. 下图表示用于11.5节讨论的联想记忆的一个人工神经网络。如果初始模式中至少有3个神经元兴奋，而中央神经元处于抑制状态，那么它与怎样的稳定布局相关联？如果初始模式中只有两个相对的周边神经元兴奋，那么将会发生什么情况？



45. 设计一个用于联想记忆（11.5节所讨论的）的人工神经网络，它由一个神经元矩形队列组成，要移动到这样的稳定模式：其中一个纵列的神经都处于兴奋状态。

46. 调整图11-18所示的人工神经网络中的权值和阈值，使其在两个输入相同（全为0或全为1）时输出为1，两个输入不同（一个为0另一个为1）时输出0。

47. 画一个与图11-5类似的图，表示把代数表达式$7x+3=3x-5$简化为$x=-2$的过程。

48. 扩展上题的答案，说明解题时控制系统可遵循的其他路径。

49. 画一个与图11-5类似的图，表示从初始事实“Polly is a parrot”、“A parrot is a bird”以及“All birds can fly”中得到结论“Polly can fly”的推理过程。

50. 与上题中的句子不同，有些鸟不会飞，如鸵鸟或是折了翅膀的鸟。但是，要建立一个演绎推理系统，其中把对陈述“All birds can fly”的所有例外都明确列出，看来并不合理。那么，我们人类如何确定一只鸟是能飞还是不能飞呢？

51. 详述句子“I read the new tax law”在不同上下文中的不同含义。

52. 说明怎样能够把从一个城市旅行到另一个城市的问题设计成一个产生式系统。什么是状态？什么是产生式？

53. 假定你要执行A、B和C3个任务，它们可以以任意次序执行（但不能同时）。说明这个问题如何设计成一个产生式系统，并画出其状态图。

54. 对于上一题中状态图，如果任务C一定要在任务B之前执行，那么怎样改变状态图？

55. a. 如果(i, j)用来表示“若一个列表中第i个位置的项大于第j个位置的项，则把两项交换”，其中i和j为正整数，那么下面两个序列中哪一个能更好地完成一个长度为3的列表的排序？

(1, 3)(3, 2)

(1, 2)(2, 3)(1, 2)

b. 注意，通过这种方式表示交换序列，序列能够加入子序列，然后重新结合形成新的序列。使用该方法，描述一种遗传算法，用于开发一个排序长度为10的列表的程序。

56. 假定一组机器人的每个成员都配备有一对传感器，每个传感器都能探测到正前方2m范围内的物体。每个机器人的形状都像一个圆的垃圾筒，能在任何方向移动。试设计一系列实验，

用来确定传感器装在哪里，使得制造出的机器人能成功地将一个篮球直线抛出。你的一系列实验如何与一个进化系统相比较？

57. 你做出某种决定是基于反应模式还是基于计划模式？你的回答是否依赖于问题是关于决定中午吃什么还是作出职业决策？

社会问题

下面的问题有助于你分析一些与计算领域相关的道德、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 核能、遗传工程以及人工智能领域的研究者对于他们研究成果的利用方式承担多大责任？科学家对其研究揭示的知识是否负有责任？若因此产生了意想不到的后果，怎么办？
2. 怎样区分智能和模拟的智能？你认为二者有区别吗？
3. 假定一个计算机化的专家系统因其给出好的建议而在医疗界享有盛誉。作为一个医生，在多大程度上可以让这个系统代替他（她）为病人作出治疗决定。如果医生的治疗方案与专家系统提出的治疗方案相对立，并且后来证实专家系统是正确的，那么那个医生是否应该对其不当治疗负有责任？一般说来，如果一个专家系统在某个领域内很有名，那么在多大程度上它会束缚而不是提高人类专家的判断力？
4. 许多人认为计算机的行为只不过是人类对它进行编程的结果，所以计算机不可能有自主意志。从而，计算机也不应对它的行为负责。人脑是计算机吗？人是否在出生的时候就事先被编程好了？人是否被他所处的环境编程？人是否要对自己的行为负责？
5. 是否有这样一些手段，科学即使能够去做，也不应当去做？例如，如果有朝一日可以造出感知和推理能力能与人类匹敌的机器，那么建造这样的机器是否恰当？这种机器的出现会带来什么样的问题？今天其他一些科学领域的进展正在引发哪些问题？
6. 历史上有许多例子表明，科学家、艺术家的创作活动受其所处时代的政治、宗教及其他社会因素的影响。这样的一些因素以何种方式影响着当今的科学成就？特别在计算机科学领域情况如何？
7. 当今，技术的进步导致一些人的工作变得多余，许多文化至少应担负起一定的责任来帮助对这些人进行再教育。随着技术使我们越来越多的能力变得多余，社会应当或能够做些什么？
8. 假定你收到一张计算机处理的费用为$0.00的账单，你该怎么办？假定你置之不理，30天后你又收到第二张$0.00的催款通知单，你该怎么办？假定你依然不理睬，而30天后你又收到了一张$0.00的催款通知单，而且还有提示，若不及时付款，将诉诸法律。谁将对此负责？
9. 是否有这样的情况，你会把个性与个人电脑联系在一起？计算机好像在施行报复或者固执难缠？计算机是否曾令你抓狂？对计算机恼火和对计算机所做的结果恼火有什么不同？计算机和你生过气吗？你与别的东西，如汽车、电视机、圆珠笔，有过类似的关系吗？
10. 根据你对上面问题的回答，人在多大程度上会把一个实体的行为与智能和意识的存在联系起来？在多大程度上，人应当做这样的关联？对于一个智能实体来说，是否可能用有别于其他行为的方式来展现它的智能？
11. 许多人觉得，能通过图灵测试并不意味着机器有智能。一个论点是，智能的行为本身并不意味智能。而进化论的基础是适者生存，这就是一种基于行为的测试。是否进化论意味着智能行为是智能的前身？机器能通过图灵测试，是否意味着它们正变得有智能？

12. 医疗手段已经取得了很大的进步，人体的许多器官现在都能用人造器官或者捐赠人的器官来替代。可以设想，终究有一天连大脑也能换。如果这变成现实，会产生什么样的道德问题？如果一个病人的神经细胞被人造神经细胞一点点换掉，那个病人还是同一个人吗？那个病人会觉察到有什么不同吗？那个病人还算人吗？
13. 一台汽车中的GPS能够提供语音提示，通知驾驶员转向及进行其他操作。当驾驶员犯错时，GPS会自动作出调整，并在不带任何不当情绪的情况下指导驾驶员返回正确路线。你是否认为当驾驶员要驾车去一个未知目的地时，GPS可以减小驾驶员的压力？GPS又会从哪些方面给驾驶员带来压力呢？

课外阅读

Banzhaf, W., P. Nordin, R. E. Deller, and F. D. Francone. *Genetic Programming: An Introduction.* San Francisco, CA: Morgan Kaufmann, 1998.

Lu, J. and J .Wu. *Multi-Agent Robotic Systems*. Boca Raton, FL: CRC Press, 2001.

Luger, G. F. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, 5th ed. Boston, MA: Addison-Wesley, 2005.

Mitchell, M. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1998.

Negnevitsky, M. *Artificial Intelligence: A Guide to Intelligent System*, 2nd ed. Boston, MA: Addison-Wesley, 2005.

Nilsson, N. *Artificial Intelligence : A New Synthesis*. San Francisco, CA: Morgan Kayfmann,1998.

Nolfi, S. and D. Floreano, *Evolutionary Robotics*. Cambridge, MA: MIT Press, 2000.

Rumelhart, D. E. and J. L. McClelland. *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986.

Russell, S. and P. Norvig. *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Prentice- Hall, 2009.

Shapiro, L. G. and G. C. Stockman. *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 2001.

Shieber, S. *The Turing Test*. Cambridge, MA: MIT Press, 2004.

Weizenbaum, J. *Computer Power and Human Reason*. New York: W. H. Freeman, 1979.

第12章

计 算 理 论

本章将讨论计算机科学的理论基础。从某种意义上说，正是本章所讨论的内容为计算机科学奠定了其真正的学科地位。尽管本质上有些抽象，但该知识体系已经有许多非常实际的应用。具体来说，我们将讨论有关编程语言能力的内在问题，以及如何通过它来构建广泛用于因特网通信中的公钥加密系统。

本章内容

本章要讨论的是有关计算机能做什么以及不能做什么的问题。我们将看到，一种称为图灵机的简单机器如何被用来确定机器可解问题与机器不可解问题之间的界线。我们还将确定一个特定的问题，就是停机问题，这个问题的解决超出了算法系统的能力，所以也就超出了当今乃至未来计算机的能力。而且，我们会发现，即使在机器可解的问题中，仍然存在一些复杂的问题，从任何实际的角度来看还是不可解的。最后要讨论的是，复杂性领域内的知识如何被用来构建公钥加密系统。

12.1 函数及其计算

本章的目的在于研究计算机的能力。我们要理解机器能做什么和不能做什么，以及机器要实现其全部潜能需具备要哪些特征。这里，就从计算函数的概念开始进行讨论。

从数学意义上讲，**函数**（function）是一组可能的输入值和一组可能的输出值之间的映射关系，它使每个可能的输入被赋予单一的输出。例如，将以码为度量单位转化为以米为度量单位。如果是同样的距离，每次用码作为单位度量与用米作为单位度量的结果之间存在着对应关系。再如排序函数，该函数对每个输入的数值表都赋予了一个输出表，而输出表的数据项与输入表一样，但是按照升序排列。还有一个例子就是加法函数，该函数的输入是一对数值，而输出值代表的是该对输入值之和。

对于一个给定的输入，确定其具体的输出值，这样一个过程称为*函数的计算*。对函数进行计算的能力非常重要，因为正是通过对函数进行计算，问题才能得到解决。为了解决一个加法问题，就必须计算加法函数；为了对表进行排序，则必须计算排序函数。因此，计算机科学的一个基本问题就是要找到一种技术，并用这种技术来计算用于求解问题的函数。

递归函数理论

没有什么比被告知某些事不能做更让人跃跃欲试。一旦研究人员开始确定一些不可解的问题，也就是说找不到解决问题的算法，就会有另外一些人开始研究这类问题，试图理解问题的复杂性。今天，递归函数理论学科就成了这样一个研究领域，并且，很多人已经认识到了这种超难问题。事实上，正如数学家开发出数字系统来揭示超越无限的“量化”级别一样，递归函数理论学家也揭开了问题空间内的多级复杂性，这些问题已经超出了算法的能力。

例如，考虑下面这样一个系统，其中，函数的输入和输出能预先确定，并记录在一个表中。每当需要函数的输出时，我们只需查找表中的给定输入，就能找到所需的输出。这样一来，这个函数的计算就简化为表的查找过程。这样的系统比较方便，但功能有限，这是因为许多函数不可能完全表示成表格形式。如图12-1所示的例子，例中展示的函数要将以码为单位的量值转化为以米为单位的量值。因为可能的输入/输出对是无限的，所以这个表注定是不完整的。

码 （输入）	米 （输出）
1	0.9144
2	1.8288
3	2.7432
4	3.6576
5	4.5720
⋮	⋮

图12-1 显示将码量度转化为米量度的函数的尝试

计算函数的一个比较有效的方法是遵循代数式所提供的方向，而不是试图将所有可能的输入/输出组合显示在表中。例如，可以用代数式

$$V=P(1+r)^n$$

来表示怎样计算一个为P的投资额（年复利率为r）在n年后的金额。

但是，代数式的表达能力也有它的局限性。有些函数，它的输入/输出关系太过复杂，以致不能用代数运算来描述。这样的例子包括三角函数，如正弦和余弦函数等。如果要计算38°的正弦值，则可能会画出相应的三角形，测出它的边长，然后计算所求的比率，而这样的一个过程就不能表示为对数值38的代数运算。用袖珍计算器来计算38°的正弦值也是比较费劲的。实际上，对38°的正弦值而言，必须利用较复杂的数学方法来得到一个非常接近的近似值，并将此作为答案。

于是可以看出，当考虑的函数越来越复杂时，我们不得不应用更为强大的方法来计算它们。然而，不管函数的复杂性如何，我们是否总能找到一个系统来计算它们？答案是否定的。有一个结论令人难受,那就是存在这样的一些函数，它们过于复杂以致找不到定义好的、一步一步的过程来根据输入值确定其输出值。结果，这些函数的计算就超出了任何算法系统的能力范围，这样的函数就称为不可计算的。而有些函数，如果可以依据它们的输入值，通过算法来确定其输出值，就称其为**可计算的**（computable）。

在计算机科学中，可计算函数与不可计算函数之间的区别很重要。这是因为，机器只能执行由算法描述的任务，所以可计算函数的研究最终是对机器能力的研究。如果我们能够确定这样的

能力，即允许机器计算所有可计算函数的能力，并造出具有这些能力的机器，那么就可以确信，所建造的机器的就如我们所设想的一样强大。同样，如果发现一个问题的解决方案需要计算一个不可计算函数，那么可以得出这样的结论：该问题的求解超出了机器的能力范围。

问题与练习

1. 举出一些函数，要求它们能完全由表格形式表示。
2. 举出一些函数，要求其输出可以描述为包括其输入的一个代数表达式。
3. 举出一个函数，要求不能用代数式来描述。那么你的这个函数是否仍然为可计算的？
4. 古希腊数学家用直尺和圆规画图形。他们借此探索出一些方法来找一条直线的中点，构建一个直角，以及画一个等边三角形。然而，他们的“计算系统”不能完成的“计算”是什么？

12.2 图灵机

在理解机器的能力以及它的局限性的工作中，许多研究人员已经提出并研究了各种不同的计算设备。其中之一就是图灵机，它是由阿兰·图灵于1936年提出来的，而在今天，它仍然被用作研究算法处理能力的一种工具。

12.2.1 图灵机的原理

图灵机（Turing machine）是由一个控制单元组成的，它能够通过一个读/写磁头对磁带上的符号进行读和写（如图12-2所示）。磁带两端可以无限延伸，并分成一个个单元，而每个单元可以包含符号的任意一个有限集合，这个集合称为机器的字母表。

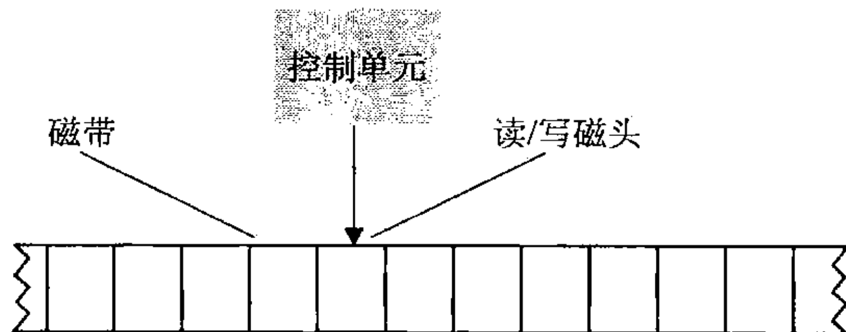


图12-2 图灵机的组成

图灵机的起源

20世纪30年代，在技术能够提供我们现在所知道的机器之前，阿兰·图灵就提出了图灵机的概念。事实上，图灵所想的是人用铅笔和纸来进行计算。图灵的目的是提供一个模型，并且利用这个模型来研究“计算过程”的局限性。在此前不久，1931年哥德尔（Gödel）发表了著名的揭示计算系统局限性的论文，并且其研究的主要精力集中在理解这些局限性上。在图灵提出他的模型的同一年（1936年），埃米尔·波斯特（Emil Post）提出了另外一种模型（现在将其称为波斯特产生式系统），他所提出的这个模型与图灵的模型有着同样的能力。作为这些早期研究人员洞察力的见证，他们的计算系统模型（如图灵机和波斯特产生式系统等）在计算机科学研究领域，仍然可以作为有价值的工具来使用。

在图灵机计算的任何一时刻，机器一定处在有限个条件中的一个，这些条件称为状态。图灵机的计算开始于一个特定的状态，称为初始状态，而停止于另一特定的状态，称为停止状态。

图灵机的计算由机器的控制单元执行的一系列步骤所组成。每一步都包括观察当前磁带单元中的符号（由读/写磁头所看到的那个），然后将符号写进这个单元，期间可能要将读/写磁头左移或右移一个单元，接下来再改变状态。要执行的确切操作是由程序决定的，程序通过机器的状态和磁带当前单元的内容来告诉控制单元做什么。

现在来考虑图灵机的一个具体的例子。为此，将机器的磁带表示成一条水平条带，并将条带分成一个个单元，且单元中可以记录机器字母表里的符号。可以通过在磁带当前单元放置一个标签来标示机器的读/写磁头的当前位置。本例中的字母包括有0、1和*。机器的磁带的样子如图12-3所示。

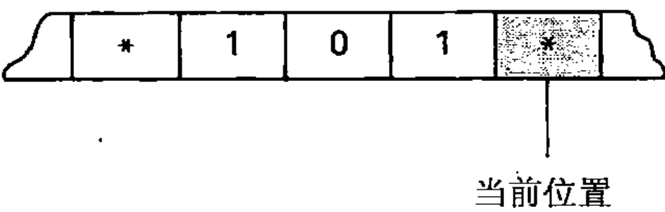


图12-3 机器的磁带的样子

磁带上的符号串可以解释为由星号分开的二进制数，那么可以看出，这个磁带包含的是值5。我们所设计的图灵机要把磁带上的这样一个值加1。更准确地说，假设开始位置是标在一串0和1右端的星号，接下来要做的是改变其左边的位模式，使其可以表示下一个较大的整数。

我们机器的状态有：START（开始）、ADD（相加）、CARRY（进位）、OVERFLOW（溢出）、RETURN（返回）以及HALT（停止）。这些状态对应的每一个操作和当前单元的内容如图12-4中的表所示。这里假设机器一直是从STRAT状态开始的。

当前状态	当前单元内容	写的值	移动方向	进入的新状态
START	*	*	左移	ADD
ADD	0	1	右移	RETURN
ADD	1	0	左移	CARRY
ADD	*	*	右移	HALT
CARRY	0	1	右移	RETURN
CARRY	1	0	左移	CARRY
CARRY	*	1	左移	OVERFLOW
OVERFLOW	(Ignored)	*	右移	RETURN
RETURN	0	0	右移	RETURN
RETURN	1	1	右移	RETURN
RETURN	*	*	不移动	HALT

图12-4 实现增加值操作的图灵机

现在把这个机器应用到图12-3所示的包含值5的磁带上。可以观察到，当处在START状态时，当前单元包含*（在本例中），图12-3中的表指示我们要重写*，并将读/写磁头左移一个单元，这时就进入了ADD状态。做完这些后，机器的情况就如图12-5所示。

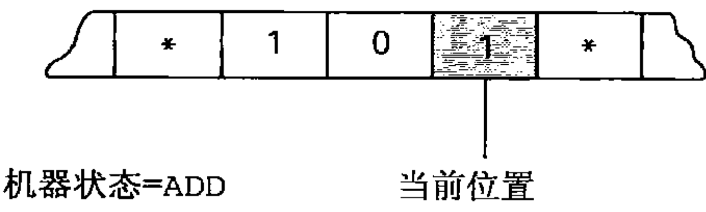


图12-5 机器状态为ADD时对应的情况

为了继续，查表看当处于ADD状态并且当前单元包含1时，机器要做些什么。图12-3所示的表告诉我们要用0代替当前单元的1，并把读/写磁头左移一个单元，这时就进入了CARRY状态。

这样一来，机器的情况就如图12-6所示。

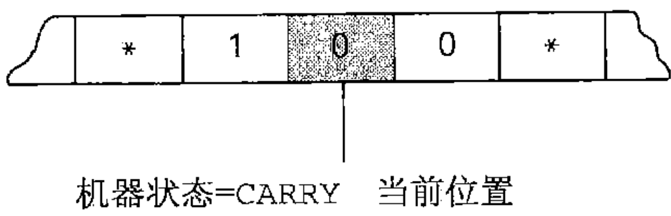


图12-6 机器状态为CARRY时对应的情况

接下来，我们再去查表，看看当机器处在CARRY状态并且当前单元包含0时要做什么。表告诉我们应该用1来代替0，并把读/写磁头右移一个单元，这时就进入了RETURN状态。做完这些后，机器的情况就如图12-7所示。

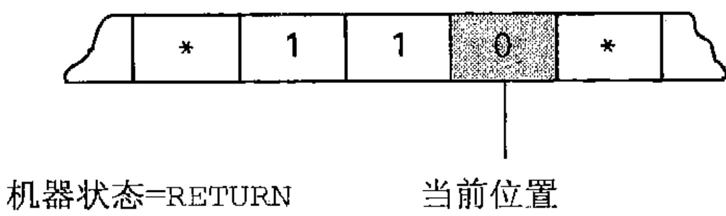


图12-7 机器状态为RETURN时对应的情况

根据这个情况，表指示我们用另一个0来代替当前单元中的0，并把读/写磁头右移一个单元，这时就保持在RETURN状态。结果，机器的情况就如图12-8所示。

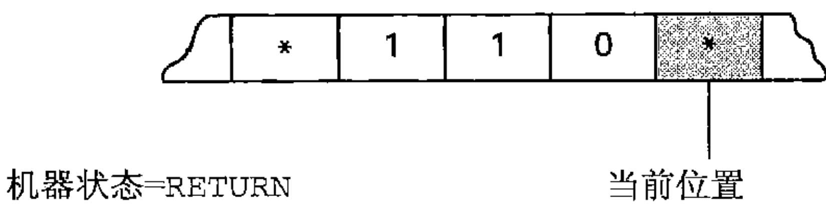


图12-8 保持RETURN状态时对应的情况

在这个时候，可以看到，表指示我们在当前单元中重写*，同时进入HALT状态。于是，机器就停止在如图12-9所示的情况（磁带上的符号就表示了所需要的值6）。

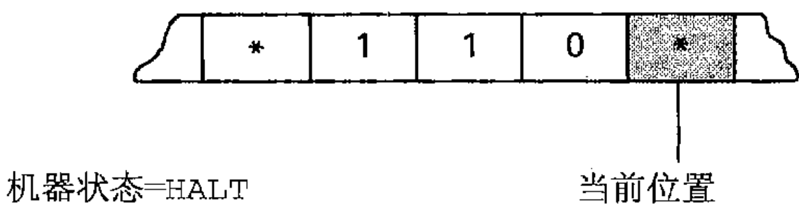


图12-9 机器状态为HALT时对应的情况

12.2.2 丘奇-图灵论题

前面例子中的图灵机可以用来计算所谓的后继函数，使用这种函数，每个非负整数输入值n的输出值为$n+1$。我们只需要把用二进制形式表示的输入值放在机器的磁带上，运行机器，直至停止，然后就可以从磁带上读取输出值。由图灵机以这种方式计算的函数称为**图灵可计算的**（Turing computable）函数。

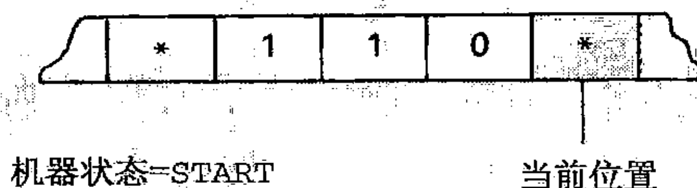
图灵猜想是指：图灵可计算函数与可计算函数是一样的。换句话说，图灵猜想，图灵机的计算能力囊括了任何算法系统的能力，或者同样也可以这么说，（与表格和代数公式这些方法形成对比）图灵机概念提供了一个环境，在此环境下，所有可计算函数的解都能够被表示。在今天，这个猜想通常被称为**丘奇-图灵论题**（Church-Turing thesis），这是为了纪念阿兰·图灵和阿隆佐·丘奇这两个人的贡献。自从图灵的最初工作以来，已经收集了许多支持这个论题的例证，

现在，丘奇-图灵论题已经被广泛接受了。也就是说，可计算函数与图灵可计算函数被认为是一回事。

这个猜想的意义就在于，它领悟到了计算机器的能力和局限性。更为准确地说，它把图灵机的能力确立为一种标准，因而其他计算系统就能够与此进行比较。如果一个计算系统能够计算所有的图灵可计算函数，那么就可以认为它的能力与任何计算系统的能力相当。

问题与练习

1. 应用本节所描述的图灵机（见图12-4），从如下的初始状态开始。



2. 描述一个图灵机，要求用一个0来替换一串0和1。
3. 描述一个图灵机，其要求是：如果磁带上的值大于0，则该值要减1；如果磁带上的值为0，则该值保持不变。
4. 请举出一个日常生活的场景，要求该场景中要有计算的活动发生。这个场景怎样与图灵机进行类比？
5. 描述一个图灵机，要求它在输入某些值时最终会停机，而在输入其他值时永不会停机。

12.3 通用程序设计语言

在第6章中，我们讨论了高级程序设计语言中的各种特性。本节中，我们要应用可计算性方面的知识来确定这些特性中哪些特性是真正必需的。我们会发现，当今的高级语言中的许多特性仅仅是增强使用的方便性，而对语言的基本功能并没有什么贡献。

我们的方法是描述一种简单的指令性程序设计语言，而这种丰富的语言足以用来表达计算所有图灵可计算函数（因此也包括所有可计算函数）的程序。因此，如果以后的程序员发现一个用这种语言解决不了的问题，那么其原因并不在于这种语言的缺陷。相反，问题就出在没有解决这个问题的算法。具有这种特性的程序设计语言称为**通用程序设计语言**（universal programming language）。

你也许会惊奇地发现，一种通用程序设计语言其实并不需要很复杂。事实上，我们将要介绍的这种语言将会非常简单。因为它是从通用程序设计语言中分离出来的需求的最小集合，所以将它称为Bare Bones（基本要素）语言。

12.3.1 Bare Bones 语言

为了表述Bare Bones语言，这里就先来考虑其他程序设计语言中的声明语句。尽管机器本身只能处理二进制位模式，并且不知道模式所表示的内容，但是利用这些声明语句程序员可以从数据结构和数据类型（如数值数组和字符串等）方面考虑问题。用来处理精巧的数据类型和数据结构的高级指令在提交给机器执行之前，必须被翻译成机器指令，这些指令操纵位模式来模拟所需的操作。

为了方便，可以将这些位模式解释成二进制符号表示的数值。这样一来，由计算机完成的所有计算都能够表示成包括非负整数的数值计算，这是有目共睹的。而且，如果要求程序员按这种方式表示算法，那么程序设计语言就能得到简化（尽管这会大大增加程序员的负担）。

由于我们开发Bare Bones语言的目标是开发出最简单的语言，所以我们将遵循这个思路。Bare Bones语言中的所有变量都考虑表示成位模式，为了方便，我们将其解释为二进制符号表示的非负整数。这样一来，一个当前被赋值为模式10的变量将包含值2，而被赋值为模式101的变量将包含值5。

利用这种约定，Bare Bones程序中的所有变量都属于同一种类型，这样一来，这种语言就不需要声明语句来描述不同变量的名字和与之相关的属性。当利用Bare Bones语言时，程序员可以在需要时只使用一个新变量名即可，这里，程序员理解的是，它是一个解释成非负整数的二进制位模式。

当然，用在Bare Bones语言中的翻译器必须能够把变量名和其他术语区分开来。要做到这一点，需要设计出Bare Bones语言的语法，以便只需通过语法就可以识别出任何术语的作用。为了达到这个目的，我们规定：变量名必须以英文字母开始，后面可以跟字母和数字（0～9）的任意组合。这样一来，字符串`XYZ`、`B747`、`abcdefghi`以及`X5Y`都能用作变量名，而`2G5`、`%o`和`x.y`就不能。

现在，让我们来考虑Bare Bones语言中的过程语句。这里有三个赋值语句和一个表示循环的控制结构语句。该语言是一种自由格式的语言，于是每条语句都以分号结束，这使得翻译器很容易将出现在同一行里的语句分割开。然而，为了增强可读性，这里仍采用的是每行只写一条语句的原则。

三条赋值语句，每条都要求改变语句中所标识的变量的内容。第一条语句可以让一个变量清零，其语法为

```
clear name;
```

其中*name*可以是任何变量名。

另外两条赋值语句的作用本质上是相反的：

```
incr name;
```

和

```
decr name;
```

同样，*name*表示任何变量名。第一条语句使标识的变量值增加1。这样一来，如果变量`Y`原来的值为5，那么执行语句

```
incr Y;
```

后，赋给变量`Y`的值就变为6。

相反，`decr`语句被用来将标识的变量值减1。一种例外的情况是，当变量的值已经为0时，这条语句将保持值不变。所以，如果变量`Y`的值为5，那么执行语句

```
decr Y;
```

后，变量`Y`所赋的值就为4。然而，如果变量`Y`的值已经为0，那么执行这条语句后，该变量的值仍为0。

Bare Bones语言只提供了一条控制结构语句，该语句由`while-end`语句对表示。语句序列

```
while name not 0 do;
 .
 .
 .
end;
```

（其中的*name*表示任意变量名）在变量*name*不为0的情况下，使位于while与end之间的任何语句或语句序列反复执行。更为准确地说，在程序执行期间遇到while-end结构语句时，所标识变量的值首先和0进行比较：如果值为0，则跳过此结构，继续执行end后面的语句；然而，如果变量的值不为0，那么就执行while-end结构中的语句序列，并且控制回到while语句，于是再进行比较。注意，程序员要担起循环控制的一部分责任，为了避免陷入无限循环，程序员必须在循环体中明确要求改变变量的值。例如，语句序列

```
incr X;
while X not 0 do;
  incr  Z;
end;
```

将会导致一个无穷的循环过程，这是因为一旦到达while语句，X的值永远不会为0。而语句序列

```
clear Z;
while X not 0 do;
  incr  Z;
  decr  X;
end;
```

最终会停止。该语句的作用是将X的初始值转移给变量Z。

可以观察出，while和end语句必须成对出现，且while语句在前。然而，while-end语句对也可以出现在被另一个while-end语句对重复执行的结构中。在这种情况中，while和end语句配对是这样实现的，即先按程序的编写形式从头到尾地对程序进行扫描，并将每条end语句与其最近的、还没有配对的前面一条while语句关联成一对。虽然在语法上并非必需的，我们通常还是采用缩进的形式来增加这种结构的可读性。

最后一个例子是图12-10中的指令序列，该序列执行的结果是将X和Y的值的乘积赋给Z，虽然有一个的副作用，即会破坏已经赋值给X的任何非零值。（由变量W控制的while-end结构起到了恢复Y的初始值的作用。）

```
clear Z;
while X not 0 do;
   clear W;
   while Y not 0 do;
      incr Z;
      incr W;
      decr Y;
   end;
   while W not 0 do;
      incr Y;
      decr W;
   end;
   decr X;
end;
```

图12-10　一个用于计算X×Y的Bare Bones程序

12.3.2　用 Bare Bones 语言编程

记住，我们提出Bare Bones语言的目的就是要研究什么是可能的，什么是不切实际的。如果要在实用的场合使用Bare Bones语言，事实证明将不太合适。另一方面，我们将很快看到，这种简单的语言达到了我们的目的，即提供了一种基本的通用程序设计语言。在这里，我们只是要说明一下如何用Bare Bones语言来表示一些基本的操作。

首先可以注意到，运用几个赋值语句的组合可以把任何值（任何非负整数）赋给一个指定的变量。例如，以下语句序列用来实现把值3赋给变量X，即先将值0赋给X，然后对其值进行三次递增操作：

```
clear  X;
incr   X;
incr   X;
incr   X;
```

程序中另一种常见的活动就是将数据从一个地方复制到另外一个地方。就Bare Bones语言而言，这就意味着我们需要能够将一个变量的值赋给另外一个变量。可以这样来实现：先将目标变量清0，然后对其进行合适次数的递增操作。事实上，我们已经看到，语句序列

```
clear  Z;
while  X  not  0  do;
  incr  Z;
  decr  X;
end;
```

把X的值转移到了Z。然而，这个语句序列还有一个副作用，即破坏了X的初始值。为了避免这一现象，可以引入一个辅助变量，先将对象的值从其初始位置转移至这个辅助变量。于是，就可以将这个辅助变量作为数据源，并从中恢复初始的变量，同时将变量的值放至所要求的目标位置上。通过这种方式，图12-11所示的语句序列实现了Today到Yesterday的转移。

```
clear Aux;
clear Tomorrow;
while Today not 0 do;
   incr Aux;
   decr Today;
end;
while Aux not 0 do;
   incr Today;
   incr Tomorrow;
   decr Aux;
end;
```

图12-11　实现指令“copy today to tomorrow”的Bare Bones语句序列

我们采用语法

```
copy  name1 to name2;
```

（这里*name1*和*name2*都表示变量名）作为一种简略的符号，用来表示图12-5所示的语句结构。这样一来，尽管Bare Bones语言本身没有明确的copy指令，但是在写程序的时候就好像有这样的指令。而这里需要理解的是，要将这种非正式的程序转化成实际的Bare Bones语言程序，必须把copy语句用其等价的while-end结构来代替，并且所使用的辅助变量名不要与程序中其他地方已经用过的名字相冲突。

12.3.3 Bare Bones 的通用性

现在就应用丘奇-图灵论题来证明我们的论断，即Bare Bones语言是一种通用程序设计语言。首先，可以看到，任何用Bare Bones语言所写的程序都能看作是对一个函数计算的指导。函数的输入包含的是程序执行前赋予变量的值，并且函数的输出包含的是程序结束时变量的值。为了计算这个函数，只要从变量的适当赋值开始执行这个程序，然后再观察程序终止时变量的值。

在这些条件下，程序

```
incr X;
```

负责计算由12.2节中图灵机例子所计算的同一个函数（后继函数）。事实上，它将X的值增加1。同样，如果将变量X和Y解释成输入，而将变量Z作为输出，那么程序

```
copy Y to Z;
while X not 0 do;
 incr Z;
 decr X;
end;
```

负责加法函数的计算。

研究者已经证明，Bare Bones程序设计语言能够用来表示计算所有图灵可计算函数的算法。如果把这与丘奇-图灵论题相结合，就意味着任何可计算函数都能由Bare Bones语言编写的程序来进行计算。这样一来，Bare Bones语言就是一种通用程序设计语言。从这个意义上讲，如果存在一个解决问题的算法，那么通过一些Bare Bones语言程序就能解决这个问题。因此，理论上可以这么说：Bare Bones语言可以用来作为一种通用程序设计语言。

之所以是从理论上讲，是因为这样一种语言当然不像第6章中介绍的高级语言那样方便。但是，每种高级语言实质上都包含有Bare Bones语言的特性，并将其作为核心。实际上，正是这个核心，才保证了每种这样的语言的通用性，而各种语言中的其他特性都是为了方便使用才包括的。

尽管像Bare Bones之类的语言在应用程序设计环境中并不实用，但在计算机科学的理论研究中还是能找到用武之地的。例如，在附录E中，将使用Bare Bones语言作为一种工具来解决第5章所提出关于迭代结构和递归结构等价的问题。事实上我们会发现，这种等价性的猜测证明是正确的。

问题与练习

1. 证明语句`invert X;`（此语句的功能是：如果X的初始值非0，那么就把X的值转化为0；如果初始值为0，那么就将该值转化为1）能够用一段Bare Bones程序段来进行模拟。
2. 证明即使简单的Bare Bones语言也包含了一些非必要的语句，如`clear`语句能利用语言中别的语句的组合来代替。
3. 证明`if-then-else`结构能够由Bare Bones语言来模拟。也就是说，用Bare Bones语言写一段程序序列，用来模拟以下语句的操作：

   ```
   if X not 0 then S1 else S2;
   ```

   其中S1和S2表示的是任意语句序列。
4. 证明：每一条Bare Bones语句都能用附录C的机器语言来表达。（所以Bare Bones语言可以作为这样一种机器的编程语言。）
5. 怎样用Bare Bones语言来处理负数？
6. 描述由下列Bare Bones程序计算的函数，假设该函数的输入由X表示，输出由Z表示。

   ```
   clear Z;
   while X not 0 do;
     incr Z;
     incr Z;
   ```

```
  decr X;
end;
```

12.4 一个不可计算的函数

现在，我们来指出一个函数，该函数属于图灵不可计算的函数，因此，依据丘奇-图灵论题，可以完全相信它在一般意义上也是不可计算的。这样一来，对这个函数的计算就超出了计算机的计算能力。

12.4.1 停机问题

我们要讨论的是与这个不可计算函数相关联的一个问题，即**停机问题**(halting problem)，(简略地说）这个问题就是要预先预测当一个程序在某些条件下开始后，是否能够终止（或者说是停止)。例如，考虑下面这个简单的Bare Bones程序：

```
while X not 0 do;
  incr X;
end;
```

如果用x的初始值0来执行这个程序，则这个循环体将不会执行，并且程序很快就可以终止。但是，如果用x的任意其他初始值来执行这个程序，那么这个循环将会永远执行下去，这样就导致了一个不可终止的过程。

于是，在这种情况下就不难得出结论：只有当x的初始值为0时，该程序的执行才会终止。然而，如果考虑更为复杂的例子，那么对程序执行行为的预测任务就变得更加复杂了。事实上，在某些情况下我们将看到，这种预测任务几乎不可能完成。但是，我们首先需要做的是规范化术语，并使想法更为精确。

我们的例子已经表明，一个程序最终能否终止就取决于其变量的初始值。这样一来，如果我们想预测一个程序的执行是否能终止，那么必须在考虑这些初始值方面要做到比较精确。为这些值所做的选择粗看起来不太习惯，但是没有关系。我们的目标是利用一种称为**自引用**(self-reference）的技术，其思想是一个对象引用自己。从“这条语句是错误的”这样的句子所表示出来的通俗的好奇心，到“所有集合的集合是否包含其自身？”这样的问题所表示的悖论，这种手法在数学上已经多次导致了令人吃惊的结果。那么，我们所要做的就是建立起一组推理的步骤，而这些步骤就类似于：“如果它是，那么它就不是；但是，如果它不是，那么它就是。”

在我们的情况中，自引用是这样来实现的，即给程序中的变量赋一个初值，而这个值就表示程序本身。为此，我们注意到，每个Bare Bones程序都是利用ASCII码，以每字节一个字符的方式编码成单个长的位模式，然后将其解释为一个（相当大的）非负整数的二进制表示。我们赋给程序中变量的初始值正是这个整数值。

现在来考虑，如果在下面这个简单程序的情形下这么做，会有什么样的结果：

```
while X not 0 do;
  incr X;
end;
```

在这里，我们想知道，如果程序开始的时候x赋予了表示程序本身的整数值（如图12-12所示)，那么执行程序后会发生什么情况。这种情况下，答案非常明显。这是因为x将会是一个非零值，

而程序就因此会陷入到死循环中。另一方面，如果用下面的程序做一个类似的试验：

```
clear X;
while X not 0 do;
  incr X;
end;
```

因为不论初始值为多少，当执行到while-end结构时，变量X的值都将是0，这样程序就会终止。

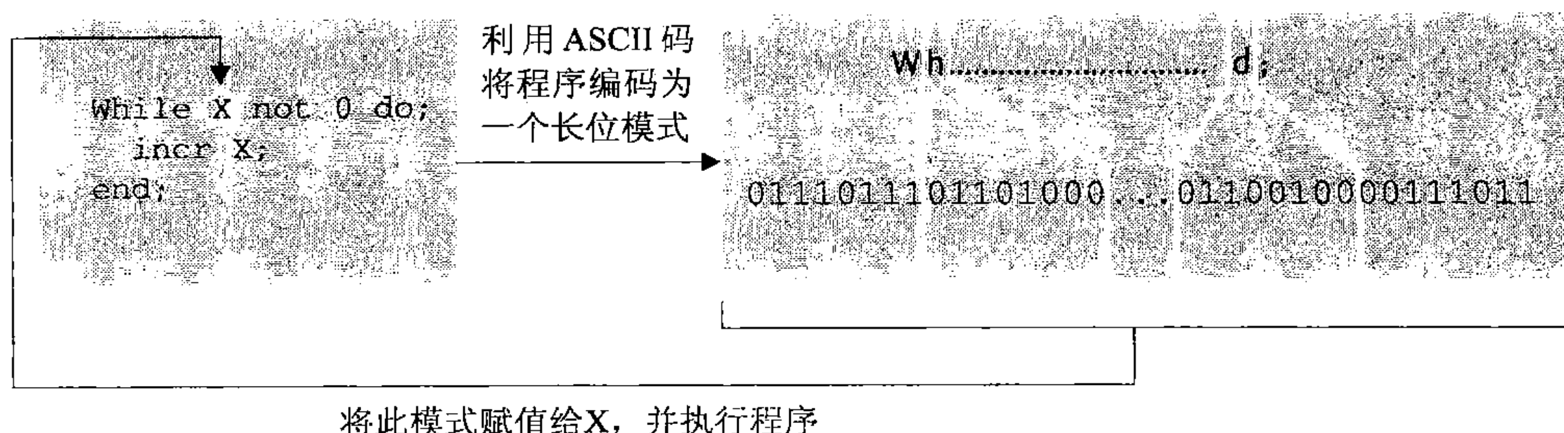


图12-12 测试一个自终止的程序

于是，可以作出以下定义：如果程序中所有的变量都用程序自身的编码表示来进行初始化，且这个程序的执行能够导致一个终止的过程，那么这个Bare Bones程序就是**自终止的**(self-terminating)。简单来说，如果一个程序以自身作为输入开始执行且能终止，那么这个程序就是自终止的。因而，这就是我们所期望的自引用。

注意，一个程序是否是自终止的可能与编写这个程序的目的无关。它仅仅是一种属性，每个Bare Bones程序要么具有这种属性，要么不具有这种属性。也就是说，每个Bare Bones程序要么是自终止的，要么就不是。

现在，可以以一种更为精确的方式来描述停机问题。这个问题就是确定Bare Bones程序是不是自终止的。我们将要看到，通常来说没有回答这个问题的算法。也就是说，当给定任何一个Bare Bones程序时，没有一个单纯的算法能够确定这个程序是不是自终止的。因此，停机问题的解决方案超出了计算机的能力。

这样的一个事实，即在我们前面的例子中看上去已经解决了停机问题，而现在却声称停机问题是不可解的，听起来有些矛盾。所以这里要暂停下来加以解释。前面例子中所用到的观察法只对那些特定的情况适用，而不能将其运用到所有的情况中去。停机问题所要求的是一种单一的、一般性的算法，并能够用在任何的Bare Bones程序中，以确定它是否是自终止的。这里，这样一个事实，即运用某些孤立的观察能力来确定某个程序是否为自终止的，并不意味着存在着一个单一的、通用的且能够适用于所有情况的方法。简而言之，我们也许能够建造出能够解决某个特定停机问题的机器，但是不能建造出一个单一的机器，使之能用来解决出现的任何停机问题。

12.4.2 停机问题的不可解性

现在，我们要来证明求解停机问题超出了机器的能力。我们的方法是要证明，解决这类问题需要一个用来计算不可计算函数的算法。所涉及函数的输入是Bare Bones程序的编码版本，其输入仅限于0和1。更准确地说，我们这样定义这个函数：表示一个自终止程序的输入就产生

输出值1，而表示一个非自终止程序的输入则产生输出值0。为了简明起见，称这个函数为停机函数（halting function）。

我们的任务就是要证明：停机函数是不可计算的。所用到的方法是“反证法”。简而言之，要证明一条语句为假，只需证明它不为真即可。于是，证明语句“停机函数是可计算的”不为真。我们的整个的论据都概括在图12-13中。

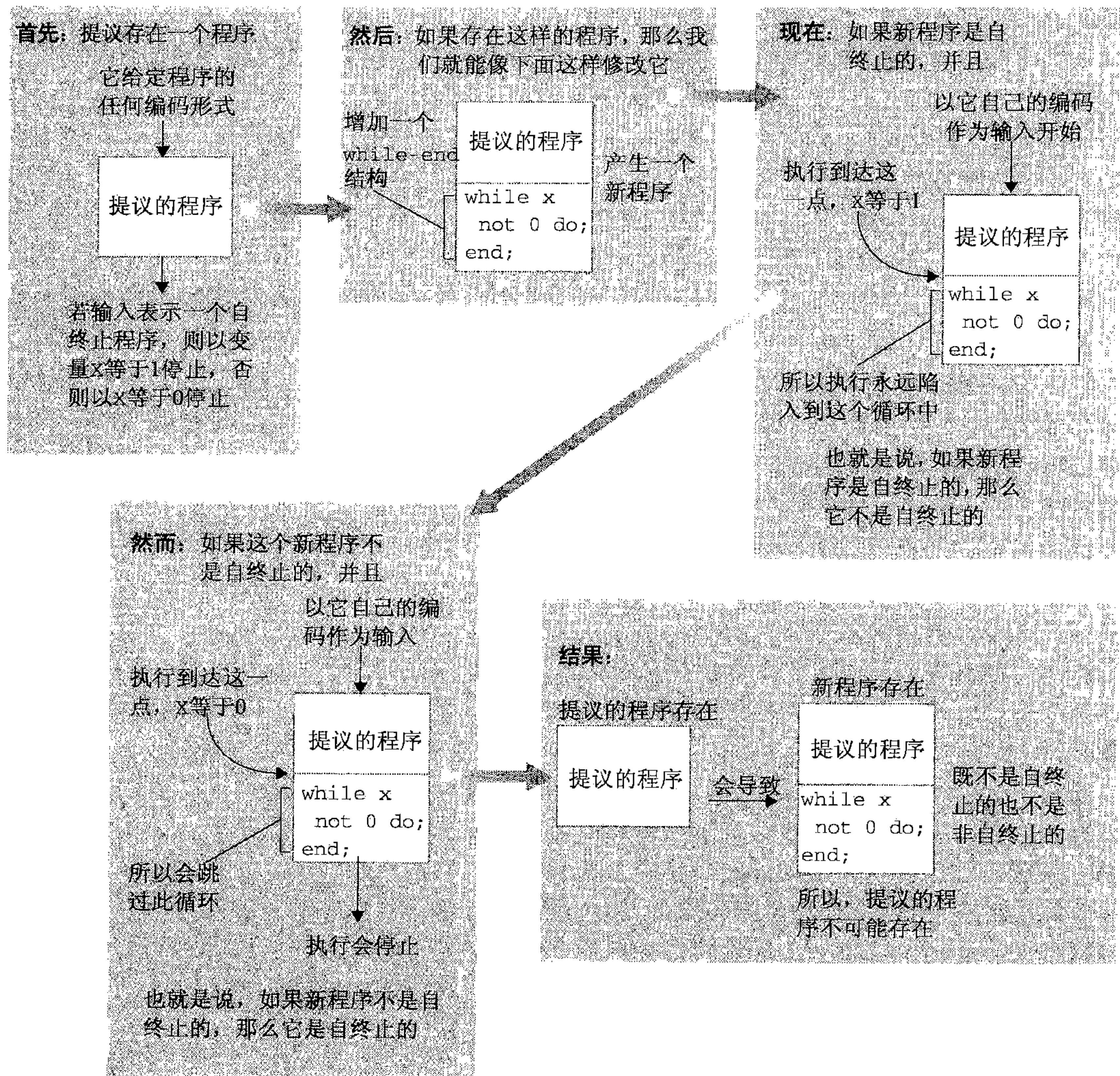


图12-13 证明停机程序的不可解

如果停机函数是可计算的，那么（因为Bare Bones语言是一种通用程序设计语言）一定存在一个能计算该函数的Bare Bones程序。换句话说，存在一个Bare Bones程序，如果它的输入是一个自终止程序的编码版本，那么它就将以输出值等于1而终止；否则就以输出值等于0而终止。

为了用这个程序，我们并不需要确认哪个变量是输入变量，而只需把程序的所有变量初始化为被测试程序的编码表示即可。这是因为，如果一个变量不是输入变量，那么它的初始值本质上是不会影响到最终的输出值的。所以，可以这么说，如果停机函数是可计算的，那么就存在下面这样一个Bare Bones程序：如果其所有变量都初始化成一个自终止程序的编码版本，那么它将以输出值等于1而终止；否则将以输出值等于0终止。

假设该程序的输出变量名为X（如果不是，则对变量进行简单的更名即可），我们可以在程序的最后加上以下的语句来修改这个程序，从而产生一个新程序：

```
while X not 0 do;
end;
```

而这个新程序必须要么是自终止的，要么就不是。然而，我们将会看到，它既不是自终止的，也不是非自终止的。

具体来说，如果这个新程序是自终止的，且用初始化为该程序自身的编码表示的变量来执行这个程序，那么当它执行到我们所加的while语句时，变量X将为1。（在这一点上，这个新程序与原始程序一样，如果其输入是一个自终止程序的表示，那么就会产生一个1。）在这一点，程序的执行将会始终陷入在while-end结构中，因为在这个循环中没有提供让X值递减的措施。但是，这与关于新程序是自终止的假设相矛盾，因此，我们必然得出结论：新程序不是自终止的。

然而，如果这个新程序不是自终止的，且用初始化为该程序自身的编码表示的变量来执行这个程序，那么当它执行到我们所加的while语句时，变量X就被赋值为0。（之所以会这样，是因为在该while语句之前的语句构成的原始程序，在其输入表示一个非自终止程序时，输出0。）在这种情况下，不会执行while-end结构中的循环，程序会停止。但是，这正是自终止程序的特性，所以，我们不得不得出结论：该新程序是自终止的。这正如早些时候我们不得不认为它不是自终止的。

概括来说，可以看出，我们遇到了程序中的一个不可能的情况，即一方面程序必须要么是自终止的，要么不是；而另一方面程序又必须既不是自终止的，又不是非自终止的。其结果是，导致这种矛盾的假设必定不成立。

我们可以得出结论：停机函数是不可计算的。因为停机问题的解决依赖于这个函数的计算，所以必然得出结论：停机问题的解决超出了任何算法系统的能力范围。这种问题被称为**不可解问题**（unsolvable problem）。

最后，把刚才讨论过的内容与第10章中的思想联系起来。第10章中一个非常基本的问题就是计算机器的能力是否包含智能本身所需要的能力。回想一下，机器只能解决使用算法可解的问题，而现在已经发现有些问题使用算法无法解决。因此，问题就在于人类的大脑是否包含了比执行算法过程更多的东西？如果没有，那么我们在这里所确定出的局限性，也就是人类思想的局限性。不必说，这是一个极具争议的问题，有时也是情绪方面的问题。例如，如果人的大脑只不过是编程过的机器的话，那么可以推断出，人类就不再拥有自由的意志。

问题与练习

1. 下面的Bare Bones程序是自终止的吗？请解释你的答案。

```
incr X;
decr Y;
```

2. 下面的Bare Bones程序是自终止的吗？请解释你的答案。

```
copy X to Y;
incr Y;
incr Y;
while X not 0 do;
```

```
    decr X;
    decr X;
    decr Y;
    decr Y;
  end;
  decr Y;
  while Y not 0 do;
  end;
```

3. 下面的场景有什么不对?

   在某个社区里，每个人都拥有自己的房子。社区的房屋油漆工声称，他要为社区内的所有房屋进行涂漆，但是只针对那些没有被屋主自己涂过的房屋。（提示：谁来涂油漆工的房屋？）

12.5 问题的复杂性

在12.4节中，已经讨论过问题的可解性。本节我们关注的问题是一个可解的问题是否有一个实际解。我们将会发现，有些问题在理论上是可解的，但由于过于复杂，从实际的观点来看，它们是无解的。

12.5.1 问题复杂性的度量

我们首先回到5.6节中的关于算法效率的研究。在那里，我们用大写的希腊字母Θ来标记，并根据算法执行所需的时间来对其进行分类。我们发现，插入排序算法属于$\Theta(n^2)$这一类，顺序查找算法属于$\Theta(n)$，而二分查找算法则属于$\Theta(\lg n)$。现在，利用这个分类系统来帮助我们确定问题的复杂性。我们的目标是开发出一种分类系统，使其能告诉我们哪些问题比另一些问题更为复杂，并最终确定出哪些问题太过复杂，以致实际上不能解。

我们现在的研究之所以是基于算法效率的知识，其原因在于希望从解题的复杂性角度来衡量一个问题的复杂性。我们认为：简单问题有简单的解法；复杂的问题就没有简单的解法。可以注意到这样一个事实，一个问题有一个复杂的解并不一定意味着该问题本身很复杂。毕竟，一个问题可以有许多解，而其中的某个解必然会复杂些。所以，如果要确定一个问题本身很复杂，那么就需要证明它的所有解都不简单。

在计算机科学领域中，让人感兴趣的问题就是那些机器能够解的问题。这些问题的解都明确地表示为算法。所以，问题的复杂性取决于解决该问题的算法的特性。更为准确地说，解决一个问题的最简单算法的复杂性可以被认为是该问题本身的复杂性。

但是，如何来衡量一个算法的复杂性呢？遗憾的是，术语**复杂性**（complexity）有着不同的解释。一种解释就涉及一个算法中所包含的判定和分支数量。如果按照这种理解，那么一个复杂的算法将会有着盘根错节的判定和分支。这种解释也许能够和软件工程师的观点相一致，软件工程师对与算法发现和表示相关的问题感兴趣，但是这并没有获得从机器的观点所看到的复杂性的概念。机器在选择下一条要执行的指令时，其实并没有做实际的判断工作，它只是一遍一遍地遵循机器周期，每次执行的都是程序计数器所给出的指令。所以，机器能够执行一组看上去很杂乱的指令，而事实上，它就像在执行一串简单有序的指令那样轻松。所以，复杂性的解释倾向于度量一个算法在表示中所遇到的难度，而不是算法本身的复杂性。

从机器的角度来看，能够更为准确地反映算法复杂性的一种解释是，要度量执行这个算法时所必须完成的步骤数。注意，这个数目与写好的程序中所出现的指令数目是不一样的。其循环体只有一条语句，但是其控制要求这个循环体执行100次的循环，在它被执行时，就相当于执行100条指令。所以，这样一个例程被认为要比一串50条分开写的语句更为复杂，尽管后者在书写形式上显得更长。复杂性最终与机器在执行一个解法时所花的时间，而不是与用来表示解的程序的大小相关。

所以，如果一个问题的所有解都需要大量的时间，那么就认为这个问题是复杂的。这种复杂性的定义称为**时间复杂性**（time complexity）。在5.6节中介绍算法效率时，我们已经间接地遇到了时间复杂性这个概念。毕竟，对算法效率的研究就是对算法时间复杂性的研究，只是两者是对立的。也就是说，“较高的效率”等于“较低的复杂性”。所以，从时间复杂性的角度看，在解决一个表单搜索问题时，顺序查找算法（属于$\Theta(n)$）要比二分查找算法（属于$\Theta(\lg n)$）更为复杂。

现在，我们运用算法复杂性方面的知识来获得一个确定问题复杂性的方法。在解一个问题时，如果存在一个算法，其时间复杂性为$\Theta(f(n))$，并且解决该问题的其他算法没有比这更低的时间复杂性，那么我们就定义这个问题的（时间）复杂度为$\Theta(f(n))$，这里$f(n)$是n的某个数学表达式。也就是说，一个问题的（时间）复杂性定义为该问题的最优解的（时间）复杂性。但是，找到一个问题的最优解和确认该解为最优解本身往往就是一个难题。在这样的情况下，大Θ标记的变种，称为**大O标记**（big O notation），被用来表示对一个问题的复杂性的了解程序。更为准确地说，如果$f(n)$是n的某个数学表达式，并且如果一个问题能够被属于$\Theta(f(n))$这一类的算法所解决，那么我们就说，这个问题是属于$O(f(n))$这一类的。这样一来，如果说一个问题是属于$O(f(n))$的，那么也就意味着这个问题有一个复杂性属于$\Theta(f(n))$的解，但是它可能还有更优解。

对查找和排序算法的研究告诉我们，一个长度为n的表（我们只知道表预先已经排序好）的查找问题是属于$O(\lg n)$的，这是因为二分查找算法能够解决这个问题。而且，研究人员已经证明，查找问题确实是属于$\Theta(\lg n)$的，所以二分查找算法就代表了这个问题的最优解。相反，我们知道，对一个长度为n的表（这时是不知道表中原始值的分布情况）的排序问题就属于$O(n^2)$，这是因为是用插入排序算法来解决这个问题的。然而，知道排序问题是属于$\Theta(n\lg n)$，这就告诉我们，插入排序算法不是最优解（从时间复杂性的角度看）。

排序问题的一个更好的解决办法是归并排序算法。其方法是将表的一些较小的、排序过的部分归并成较大的、排序好的部分，然后再进行归并，得到更大的排序过的部分。每次归并过程都是利用在介绍顺序文件时所遇到的归并算法（如图9-15所示）。为了方便，再用图12-14来表示，而这次的情况是归并两个表。完整的(递归)归并排序算法可由图12-15中所示的MergeSort过程来表示。当要求对一个表排序时，这个过程首先检查被排序的表，看其是否少于两个数据项：如果是，则该过程的任务已经完成；如果不是，则这个过程将表分成两部分，再请求过程MergeSort的另外一个副本对这两部分进行排序，然后将这些排序好的片段合并在一起，这样就得到最后的排序过的表。

为了分析这个算法的复杂性，首先来考虑在合并一个长度为r的表和一个长度为s的表时，必须要在表的数据项之间进行比较的次数。归并过程是这样进行的：重复地对一个表中的数据项与另一个表中的数据项进行比较，然后将两者中的“较小项”放入到输出表中。这样一来，每做一次比较，那么还要考虑的（也就是未比较的）数据项的数目就要减1。由于开始时只有$r+s$个数据项，那么我们就可以得出结论：这两个表的归并过程所包含的比较次数不会多于$r+s$次。

```
procedure MergeLists (InputListA, InputListB, OutputList)
if (两个输入表为空) then (Stop, OutputList为空)
if (InputListA为空)
  then (声明它是饥饿的)
  else (声明它的第一项为当前项)
if (InputListB为空)
  then (声明它是饥饿的)
  else (声明它的第一项为当前项)
while (两个输入表都不是饥饿的) do
   (把较小的当前项放入OutputList;
    if (该当前项是对应输入表的最后一项)
          then (声明该输入表是饥饿的)
          else (声明该输入表的下一项为该表的当前项)
   )
从未完的输入表中的当前项开始，剩下的项复制到OutputList。
```

图12-14 用来合并两个表的过程MergeLists

```
procedure MergeSort (List)

if (List有多项)
  then (应用过程MergeSort来对List的前半部分进行排序;
          应用过程MergeSort来对List的后半部分进行排序;
          应用过程MergeLists合并List的前半部分和后半部分生成一个已排序的List
        )
```

图12-15 实现为过程MergeSort的归并排序算法

现在来考虑完整的归并排序算法。它是通过这样的方式来处理对一个长度为n的表的排序工作，即将最初的排序问题简化为两个相对较小的问题，每个问题是要对一个长度约为$n/2$的表进行排序，接下来再对这两个问题进行分割，使其成为4个对长度约为$n/4$的表进行排序的问题。这种分割过程可以由图12-16中的树结构来概括，图中树的每个节点表示的递归过程中的一个问题，并且一个节点下面的分支表示的是从这个父节点衍生而来的更小的问题。所以，我们可以发现，将树中各个节点上发生的比较次数加起来，就得到整个排序过程中所发生的总的比较次数。

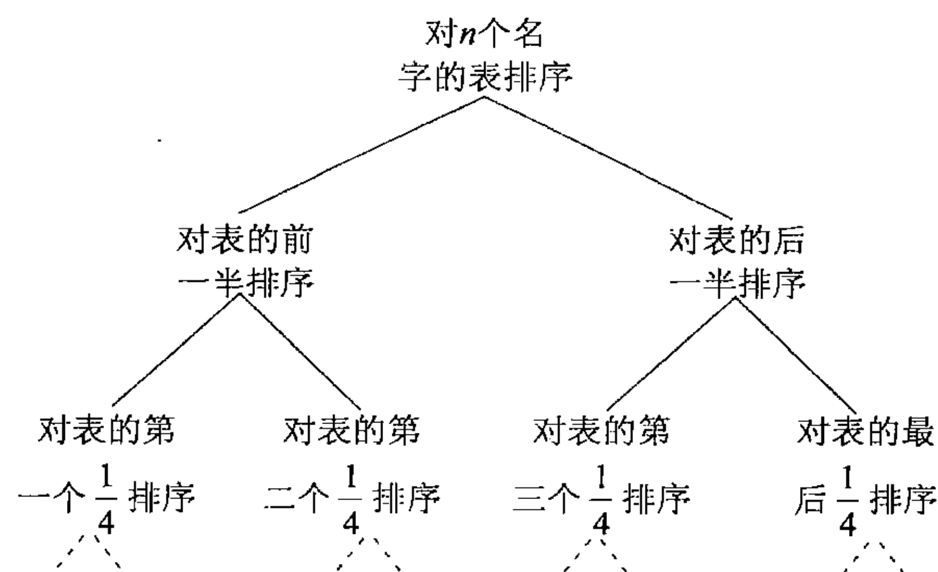


图12-16 由归并排序算法产生的问题的层次结构

首先，我们来确定树的每层上所进行比较的次数。可以看出，出现在树的任意一层的每个节点，其任务都是对初始表的一个特定段进行排序。这个工作由归并过程来完成，因此正如我们已经指出过的，所要求的比较次数不会多于该表段中的数据项的数。因而，树的每一层所需的比较次数不会多于该表段中的数据项的总数，而且，因为树中所给定的一个层的段表示的是初始表所分割的部分，因而这个总数不会比初始表的长度大。因此，树的每一层所包含的比较

次数都不会多于n。（当然，最底层所包含的排序表的长度小于2，因而根本就不需要比较了。）

现在来确定树中的层数。为此，可以看到，把问题分割成更小的问题这个过程一直进行到所得到表的长度小于2为止。这样一来，树中的层数就由分割的次数所确定，从值n开始，反复除以2，直到其结果不大于1，那么这个次数就是$\lg n$。更为准确地说，树中所涉及的比较层数不多于$n\lceil \lg n \rceil$，这里，标记$n\lceil \lg n \rceil$表示的是将$\lceil \lg n \rceil$的值向上取整。

最后，把树中每层所做的比较次数乘以涉及比较操作的层数，这样就得到了在对长度为n的表进行排序时，归并排序算法所做的总的比较次数。可以确定，这个次数不大于$n\lceil \lg n \rceil$。因为$n\lceil \lg n \rceil$对应的图形与$n\lg n$对应的图形在形状上大致一样，我们就可以得出结论：归并排序算法是属于$O(n\lg n)$的。把这个结论与研究人员告诉我们的排序问题的复杂性为$\Theta(n\lg n)$这个事实相结合，这就意味着归并排序算法代表了排序问题的一个最优解。

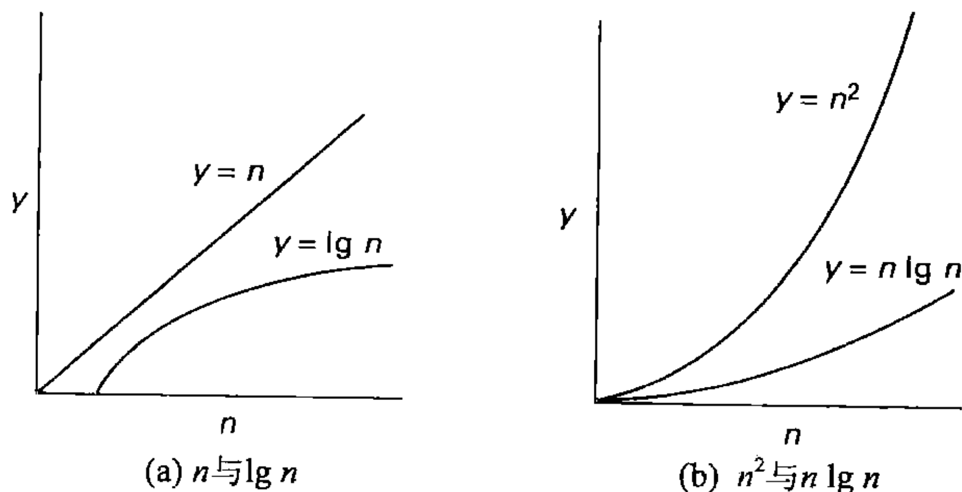
空间复杂性

除了从时间的角度来度量复杂性，还有一种方法就是通过度量所需的存储空间来衡量复杂性，我们将这种度量方法称为**空间复杂性**（space complexity）。也就是说，一个问题的空间复杂性是由解决该问题所需的存储空间的数量决定的。文中我们已经看到，一个有n个数据项的表的排序复杂性是$O(n\lg n)$。而这个问题的空间复杂性将不超过$O(n+1)=O(n)$。要知道，利用插入排序对一个有n个数据项的表进行排序，需要存放表本身的空间，还要加上用来存放临时数据项的空间。这样一来，如果要对越来越长的表进行排序，那么将会发现，每个任务所需的时间比所需的空间增长要快得多。事实上，这是一个很常见的现象。因为利用空间也要花费时间，所以一个问题的空间复杂性永远不会比它的时间复杂性增长得更快。

通常会在时间复杂性和空间复杂性之间做出一些折中。在某些应用场合中，为了方便会事先进行某些计算，并将计算结果以表格的形式存放起来，这样一来，在需要时就能通过表格很快地检索到。这样一种“查表”技术实际上是通过表格所需的额外空间的代价来换取获取数据所需时间的减少。另一方面，通常用数据压缩来减少对存储空间的需求，其代价是数据压缩和解压缩所需要的额外时间。

12.5.2　多项式问题与非多项式问题

假设$f(n)$和$g(n)$是数学表达式。如果要说$g(n)$是受$f(n)$约束的，那么这就表示当把这些表达式用在越来越大的n值上时，$f(n)$的值最终将会大于$g(n)$的值，并且对所有更大的n值，$f(n)$都将大于$g(n)$。换句话说，如果$g(n)$受$f(n)$的约束，那么也就意味着对于“较大”的n值，$f(n)$的图像将会在$g(n)$的图像之上。例如，表达式$\lg n$受表达式n的约束（如图12-17a所示），而$n\lg n$受n^2的约束（如图12-17b所示）。



(a) n与$\lg n$　　(b) n^2与$n\lg n$

图12-17　数学表达式n、$\lg n$、$n\lg n$和n^2的图像

如果一个问题是属于$O(f(n))$的，其中，表达式$f(n)$要么本身是一个多项式，要么就是受一个多项式约束，那么我们就说，这个问题是一个**多项式问题**（polynomial problem）。所有多项式问题的集合用P表示。注意，前面的讨论告诉我们，表的查找和排序问题就属于P。

说一个问题是多项式问题，这就是关于解决该问题所需时间的一种陈述。我们经常会说到，P中的问题能够在多项式时间范围解决，或者说，该问题有多项式的时间解。

确定出属于P的问题是计算机科学中非常重要的课题，这是因为，这个问题与问题是否有实际解密切相关。确实，P类之外的问题，其特征都是具有极长的执行时间，即使是对中等规模的输入也是如此。例如，考虑一个求解需要2^n步的问题。指数表达式2^n不受任何多项式的约束，也就是说，如果$f(n)$是一个多项式，那么，当增加n的值时，我们就会发现，2^n的值最终会大于$f(n)$的值。这就意味着，如果一个复杂性为$\Theta(2^n)$的算法通常会比复杂性$\Theta(f(n))$的算法效率低，因而就需要更多的时间。如果一个算法的复杂性是用指数表达式来确定的，那么就说该问题需要指数时间。

下面介绍一个具体的例子，考虑这样一个问题，即从n个人组成的群体中，列出所有可能的小组组合。因为这里可以有2^n-1种这样的组合（这里可以允许一个小组包含所有的人，但是不允许小组中没有人），所以解决此问题的任何算法必须至少有2^n-1步，这样一来，其复杂性也至少这么大。但是，表达式2^n-1作为一个指数表达式，不受任何多项式的约束。所以，随着可选人群规模的增加，对这个问题的任何解所花费的时间也变得非常庞大。

上面的分组问题，其复杂性非常大，这只是因为它的输出规模太大，而与此不同的是，存在着这样的一些问题，虽然它们最终的输出只是是或否这样简单，但是其复杂性却非常大。一个例子就是能不能回答涉及实数加法的一些语句的真实性问题。例如，对于“存在一个实数，当自身相加时就得到值6。这是真的吗？”这样的一个问题，我们就很容易得到答案，即答案为真。而对“存在一个非零实数，当自身相加时就得到值0。这是真的吗？”这样的一个问题，显然答案为假。然而，当碰到这类问题越来越多时，我们回答这些问题的能力也就会开始减弱了。如果发现自己要面对许多这样的问题，那么我们就可能尝试着求助于计算机程序。遗憾的是，回答这类问题的能力已经证明需要指数时间，所以，随着所涉及的这类问题越来越多，最终的结果是，即使是计算机也做不到以一种实时的方式给出答案。

理论上可解但不属于P的问题有着巨大的时间复杂性，这一事实使得我们得出结论：从实践角度看，这些问题本质上是不可解的。计算机科学家称这类问题为**难解型**（intractable）问题。从而，类型P成为了用来区别难解的问题与那些可能有实际解的问题之间的一个重要分界线。因此，对类型P的理解已经成为了计算机科学领域的一个重要研究内容。

12.5.3 NP 问题

现在来考虑**旅行商问题**（traveling salesman problem），该问题中涉及了一个旅行商，他必须要访问到不同城市的每个客户，其花费不能超出他的出差预算。所以，他的问题就是要找到一条路径（从家里出发，遍历有关的城市，然后再返回），其总长度不超过允许的里程。

这个问题的传统解决办法是这样的，以系统化的方式来考虑各种可能的路径，然后把每条路径的长度与里程的限制数相比较，直到找到一条可接受的路径，或者是把所有可能的路径都考虑到。然而，这种方法并不能产生一个多项式时间解。随着城市数目的增加，所要测试的路径数目比任何多项式增长得都要快。因此，在所涉及城市的数目很多的情况下，按照这种方式来解决旅行商问题是不实际的。

我们可以得出结论：如果要在一个合理的时间范围内解决旅行商问题，必须要找到一个更快的算法。如果存在着一个令人满意的路径，并且碰巧一开始就选择了这条路径，所提出的算

法也能很快地终止，那么这样就吊起了我们的胃口。具体来说，下面的指令序列能够很快地执行，并且也有解决这个问题的潜力：

```
取一个可能的路径，并计算其总距离
if (此距离不大于允许的里程数)
   then (宣布找到)
   else (宣布没找到)
```

然而，从技术意义上讲，这组指令不是一个算法。它的第一条指令就比较模糊，原因在于它既没有指出选择的是哪一条路径，又没有说明如何作出这样的决定。相反，它是依赖于程序执行机制的创造性来自己做决定。我们称这样的指令为不确定指令，并将包含这样语句的“算法”称为**不确定算法**（nondeterministic algorithm）。

确定性的和不确定性的

在许多情况下，一个确定性“算法”和一个不确定性“算法”之间存在着明显的界限。这种差别是非常清晰和明显的。确定性算法不依赖于执行该算法的机制的创造性能力，而不确定算法可能会依赖。例如，比较指令

走到下一个十字路口，然后向右转或向左转

与指令

走到下一个十字路口，按照站在路口的人所告诉你的，向右转或向左转

在这两种情况中，这个人在真正执行指令之前，所采取的转向行动都是不确定的。然而，第一条指令要求行人根据自己的判断来决定其转向，所以就是不确定的。第二条指令中，对行人所要转的方向就不做这样的要求，他只是被告知每一步做什么。如果有几个不同的人执行第一条指令，那么有些人会向右转，而有些人会向左转。如果有几个人执行第二条指令，并且得到同样的信息，那么他们将会朝一个方向转。这里包含了确定性“算法”和不确定性“算法”两者之间的一个重要区别。如果用同样的输入数据反复去执行一个确定性算法，那么每次都将会完成同样的操作。然而，一个不确定性算法在同样的条件下反复执行，将会产生不同的操作。

注意，随着城市数目的增加，执行上述不确定算法所需要的时间增加得相对较慢。选择一条路径的过程仅仅是产生一个城市清单，这能够在与城市数目成比例的时间范围内完成。而且，沿着所选择的路径，计算总距离所需的时间也与所访问的城市数目成正比，并且，将这个总数与里程的限定数进行比较所需的时间与城市的数目无关。于是，执行这个不确定算法所需的时间就受一个多项式约束。因此，就有可能在多项式时间内，利用一个不确定算法解决旅行商问题。

当然，这个不确定解并不能令人十分满意，因为它依赖于猜测的运气。但是，它的存在足以表明：在多项式时间内，对旅行商问题而言，存在着一个确定性的解。无论其真假与否，它都是一个尚未确定的问题。事实上，有许多这样的问题，即知道他们在多项式时间范围内执行时有不确定性解，但还没发现多项式时间内的确定性解，旅行商问题就这些问题中的一个。对于这些问题的不确定性解的这种可望而不可及的效率，使得许多人希望在某一天能找到有效的确定性解，然而，大多数人相信这些问题太复杂，以致超出了有效的确定性算法的能力范围。

一个能够在多项式时间内用不确定性算法解决的问题，称为**非确定性多项式问题**（nondeterministic polynomial problem），或者简称**NP问题**（NP problem）。习惯上把NP类问题表示成NP。注意，P中的所有问题也都属于NP问题，这是因为任何（确定性）算法都可以加上一条非确定指令而不影响其性能。

然而，正如旅行商问题所说明的，所有的NP问题是否也都属于P问题还是个尚未确定的问题。在今天，这也许是计算机科学领域里的最广为人知的未解问题。这个问题的解决将会带来重大的影响。例如，12.6节我们将讨论到，已经设计出来的加密系统其完整性依赖于解决问题所需的大量时间，这类似于旅行商问题。如果证实了对这样的问题存在着有效解，那么这些加密系统的安全性就将受到威胁。

为了解决这样的问题（即NP类问题在事实上是否等同于P类问题）而作出的努力，导致了在NP类问题中发现了一类称为**NP完全问题**（NP-complete problem）的问题。这些问题都具有这样一种特征，即任何一个问题的多项式时间解也为所有其他NP类问题提供了一个多项式时间解。也就是说，如果能够在多项式时间内找到一个（确定）算法，使其能够解决一个NP完全问题，那么这个算法就能推广到以多项式时间来解决任何其他于NP类问题。于是，NP类就和P类一样了。旅行商问题是NP完全问题的一个例子。

概括来说，可以看出，问题可以分为可解（有一个算法解）和不可解（没有算法解）两类，如图12-18所示。而且，可解问题可分为两个子类。一类是多项式问题的集合，该集合包含了有实际解的问题。另一类是非多项式问题的集合，这些问题只有当输入相对较少或者仔细选取的情况下才有实际解。最后，还有比较难理解的NP问题，至今还没有很准确的分类。

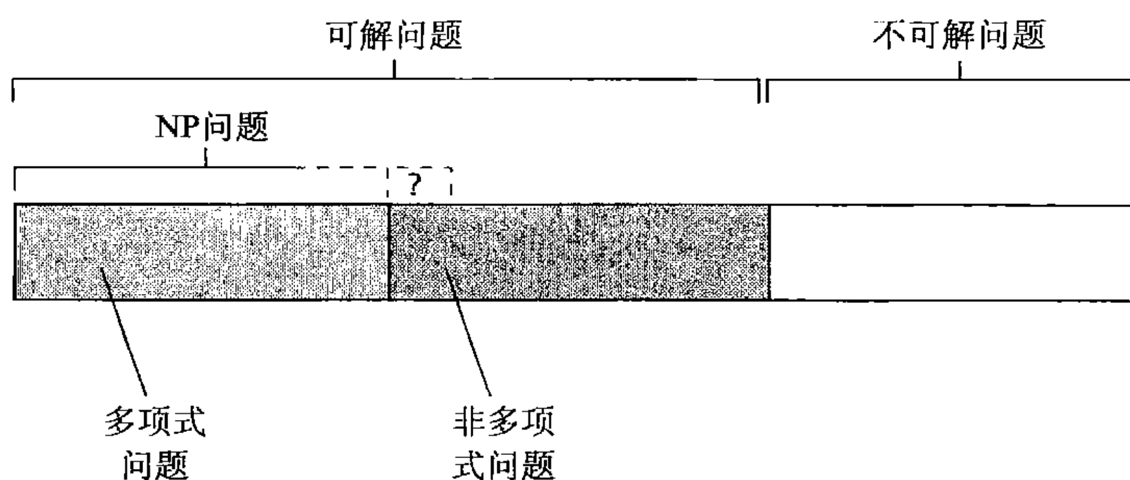


图12-18 问题分类的一个概括图

问题与练习

1. 假设一个问题能够通过一个属于$\Theta(2^n)$的算法求解，那么对这个问题的复杂性，我们能得出什么样的结论？
2. 假设一个问题能够通过一个属于$\Theta(n^2)$的算法求解，也可以通过另一个属于$\Theta(2^n)$的算法求解，那么是否一个算法总是要优于另一个算法？
3. 如果一个委员会由Alice和Bill两个成员组成，那么请列出从这个委员会的所有可能的分组。如果这个委员会是由Alice、Bill和Carol组成，那么请列出从这个委员会的所有可能的分组。如果这个委员会是由Alice、Bill、Carol和David组成的，那么分组情况又会怎样？
4. 举出一个多项式问题的例子。举出一个非多项式问题的例子。举出一个尚未被证明是多项式问题的NP问题的例子。
5. 如果算法X的复杂度比算法Y的复杂度要大，那么是否就意味着算法X一定就比算法Y难理解？请解释你的答案。

*12.6 公钥密码学

在某些情况下，一个问题难以解决这样一个事实已经不再是一个缺点，而变成了一个优点。特别有趣的一个问题是，为给定的一个整数找到它的因数，如果这样的解确实存在，那么就一定要为这个问题找到一个有效的解。例如，如果只用笔和纸，你将会发现，即使像2 173这样相对较小的数值，要找到其因数也比较花时间，而如果涉及的数大到需要用几百位数字来表示，那么即使用现在最好的分解因数的技术，这个问题还是难以解决。

对许多数学家而言，至今还没有找到一种有效的方法来确定大整数的因数，这让他们感到非常不安。然而，在密码学领域里，这种情况已经被用来产生一种对报文进行加密和解密的流行方法。这种方法称为**RSA算法**（RSA algorithm），选择这个名字是为了对该算法的发明者Ron Rivest、Adi Shamir和Len Adleman表示尊敬。这种方法，利用一组称为**加密密钥**（encrypting key）的数值对报文进行加密，并利用另一组称为**解密密钥**（decrypting key）的数值对报文进行解密。知道加密密钥的人可以对报文进行加密，但是不能对报文进行解密。只有持有解密密钥的人才能对报文进行解密。这样一来，加密密钥可以被广泛地分发而不会破坏系统的安全性。

这样的密码系统称为**公钥加密**（public-key encryption）系统，这个术语反映了用来对报文加密的密钥可以公开而不会降低系统的安全性。事实上，加密密钥通常被称为**公钥**（public key），而解密密钥则称为**私钥**（private key）（如图12-19所示）。

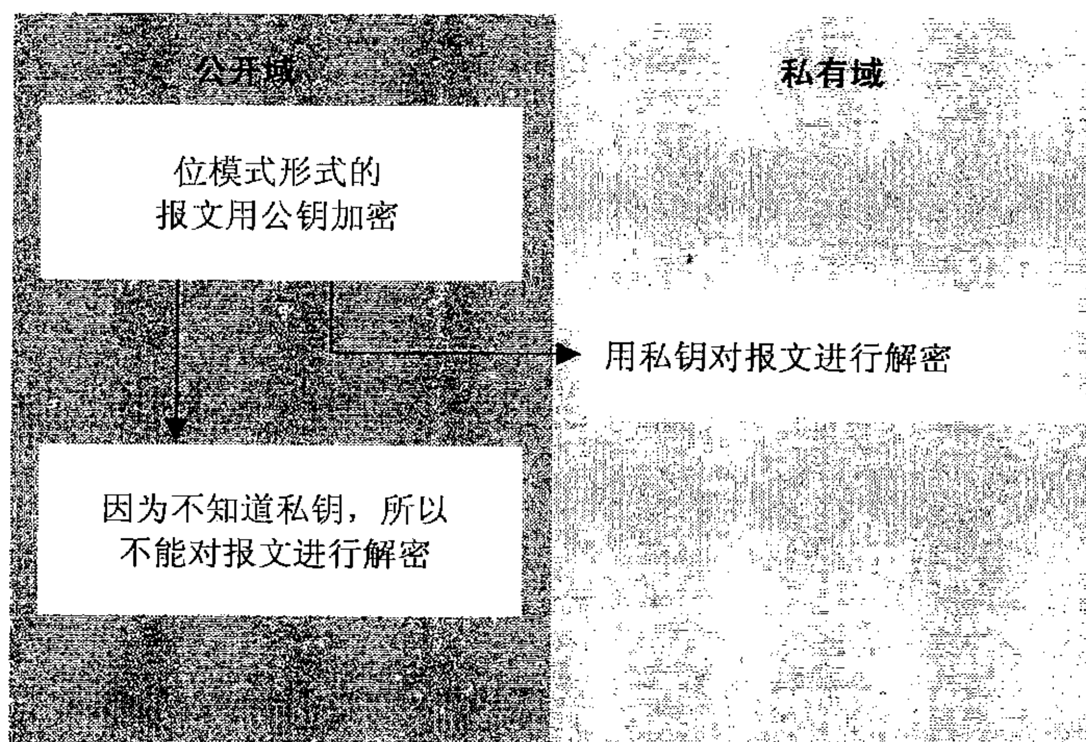


图12-19 公钥密码学

12.6.1 模表示法

为了描述RSA公钥加密系统，可以方便地采用记号$x(\text{mod } m)$来表示数值x被m除后所得的余数，这通常读作"$x \text{ mod } m$"。这样一来，9(mod 7)就得2，这是因为9÷7的余数为2。类似地，24(mod 7)为3，这是因为24÷7的余数为3；14(mod 7)为0，因为14÷7的余数为0。注意，如果x是一个0~m−1的整数，那么$x(\text{mod } m)$的值就为x本身。例如，4(mod 9)的值就为4。

借助数学知识我们知道，如果p和q是素数，m是0~pq（表示p和q的乘积）的一个整数，那么，对于任意的正整数k，就有

$$1 = m^{k(p-1)(q-1)}(\text{mod } pq)$$

尽管在这里不证明这个命题，但考虑一个例子来解释这个命题还是有必要的。于是，我们假设p

和q分别为素数3和5，并且m为整数4。那么，这个命题说，对于任意的正整数k，值$m^{k(p-1)(q-1)}$除以15（3和5的乘积）将得到余数1。具体来说，如果k=1，那么

$$m^{k(p-1)(q-1)} = 4^{1(3-1)(5-1)} = 4^8 = 65\ 536$$

正如前面所讲的，该值除以15所得余数为1。而且，如果$k = 2$，那么

$$m^{k(p-1)(q-1)} = 4^{2(3-1)(5-1)} = 4^{16} = 4\ 294\ 967\ 296$$

再将它除以15所得的余数为1。事实上，无论正整数k的取值如何，都将得到余数1。

12.6.2 RSA 公钥加密系统

现在，我们准备在RSA算法的基础上构建和分析一个公钥加密系统。首先，选出两个不同的素数p和q，其乘积用n来表示。然后，另外选出两个正整数e和d，使得对于某个正整数k，满足$e \times d = k\,(p-1)(q-1) + 1$。将值$e$和$d$分别作为加密和解密过程的组成部分。（事实上，所选取的值e和d能够满足前面的等式，这在数学上也是另一个已经证明过的事实，在这里我们不再证明。）

所以，这里选取了5个值：p、q、n、e和d。值e和n是加密密钥，值d和n是解密密钥，值p和q只是用来构建加密系统。

这里就考虑一个具体的例子来进行说明。假设将值p和q分别选取为7和13，那么n=7 × 13 = 91。而且，将值e和d分别选取为5和29，因为5 × 29=145=144 + 1 =2（7−1）（13−1）+1=2（p−1）(q−1) +1，这正是所需的。这样一来，加密密钥就是n=91和e=5，而解密密钥则为n=91和d=29。我们将加密密钥分发给想要给我们发报文的人，而解密密钥（以及p和q的值）我们自己保留。

我们现在来考虑如何对报文进行加密。为此，假设当前的报文是按照位模式（可能用的是ASCII编码或Unicode码）编码的，当将其解释为二进制表示时，这个位模式的值就小于n。（如果它不小于n，就要把报文分割成较小的段，然后分别对每段进行加密。）

假设当报文解释为二进制表示时，我们的报文表示值m。那么，这条报文的加密形式就是值$c = m^e$（mod n）的二进制表示。也就是说，加密过的报文是m^e除以n后所得余数的二进制表示。

具体来说，继续就前面的例子来进行讨论，如果某人想要用加密密钥n =91和e = 5对报文10111进行加密，他首先会看出，10111是值23的二进制表示，那么计算$23^e = 23^5$ = 6 436 343，最后，将此值除以n = 91，得到余数为4。所以这条报文的加密形式就是100，即为4的二进制表示。

为了解密一条用二进制记法表示的值为c的报文，就要计算c^d(mod n)的值。也就是说，计算c^d的值，将结果再除以n，并保留所得的余数。事实上，这个余数就是初始报文的值m，这是因为

$$\begin{aligned}
c^d \,(\mathrm{mod}\, n) &= m^{e \times d}\,(\mathrm{mod}\, n) \\
&= m^{k(p-1)(q-1)+1}\,(\mathrm{mod}\, n) \\
&= m \times m^{k(p-1)(q-1)}\,(\mathrm{mod}\, n) \\
&= m(\mathrm{mod}\, n) \\
&= m
\end{aligned}$$

正如前面所述，这里用到了 $m^{k(p-1)(q-1)}(\bmod n)=m^{k(p-1)(q-1)}(\bmod pq)=1$，以及 $m\ (\bmod\ n)=m$（因为 $m<n$）。

继续讨论前面的例子，如果收到报文是100，那么我们就能识别出这个值为4，然后计算出值 $4^d=4^{29}=288\ 230\ 376\ 151\ 711\ 744$，将此值除以 $n=91$，从而得到余数23，即为用二进制表示的初始报文10111。

概括来说，一个RSA公钥加密系统的产生过程是这样的：选取两个素数p和q，再从这两个数产生值n、e和d。值n和e被用于加密报文，即为公钥；而值n和d被用于解密报文，即为私钥（如图12-20所示）。这个系统的优势就在于只知道如何加密报文，而不能解密报文。这样一来，加密密钥n和e可以被广泛地分发。如果你的对手得到了这些加密密钥，但是他们还是不能够对他们所截获的报文进行解密，因为只有知道解密密钥的人才能对报文进行解密。

这种系统的安全性的基础在于，假定只知道加密密钥n和e，而不允许计算出解密密钥n和d。然而，确实有这样的算法能做到！一种方法是对值n进行分解因式来找到p和q，然后找一个k值，使得$k(p-1)(q-1)+1$被e整除（那么商就为d），从而确定了d。另一方面，这个过程的第一步就可能很花时间，特别是在所选的p和q很大的情况。事实上，如果p和q大到需要用几百个二进制位来表示时，那么即使用最好的分解因式算法对n进行分解，也得需要好几年的时间才能确定p和q。因此，一条加密过的报文的内容，即使其重要性已经过时很久，它的安全性仍然能得到保证。

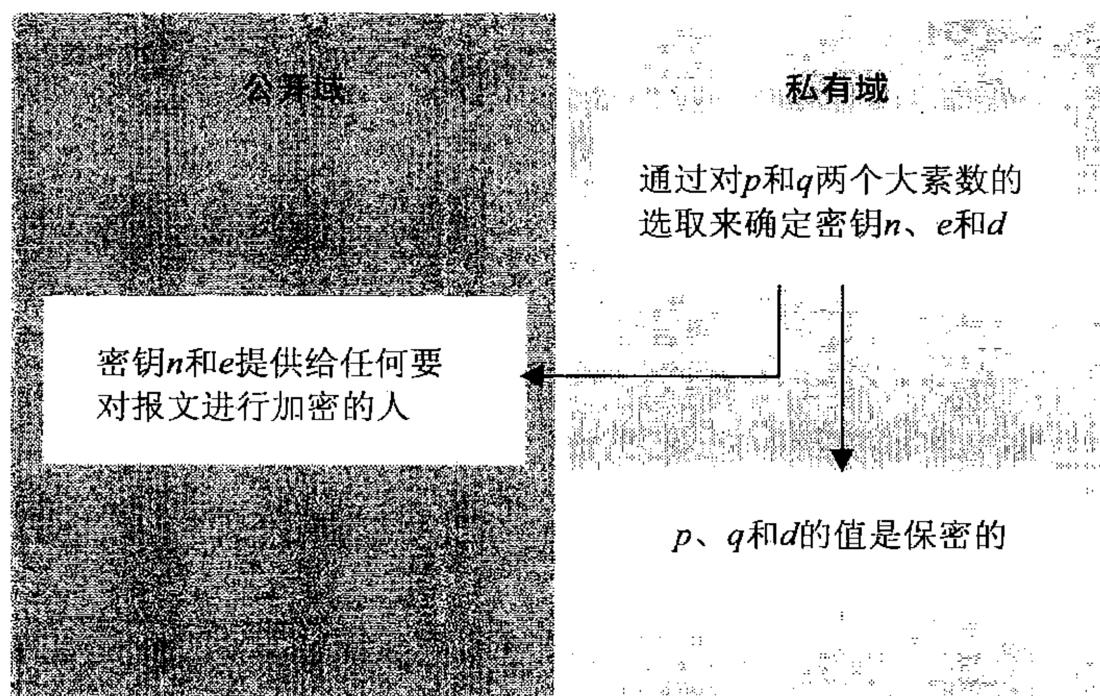


图12-20 建立一个RSA公钥加密系统

到今天，还没有人能够在不知道解密密钥的情况下，找到一种对基于RSA加密算法加密的报文进行解密的有效方法，所以，基于RSA算法的公钥加密技术广泛地用于因特网通信，以获取通信的秘密性。

问题与练习

1. 请找出66 043的因数。（本题可能比较费时，不必花费太多的时间。）
2. 用公钥$n=91$和$e=5$对消息101进行加密。
3. 用私钥$n=91$和$d=29$对消息10进行解密。
4. 在一个RSA公钥加密系统中，根据素数$p=7$、$q=19$以及$e=5$，为解密密钥n和d找出合适的值。

复习题

1. 请说明一下，如何用Bare Bones语言模拟如下结构：

```
while X equals 0 do;
 .
 .
 .
end;
```

2. 写一个Bare Bones语言程序，使得：如果变量X小于等于变量Y，则将变量Z置为1，否则将变量Z置为0。
3. 写一个Bare Bones语言程序，将变量Z置为2的X次方。
4. 根据以下每种情况写一个Bare Bones语言程序序列，实现指定的活动。
   a. 如果X的值为偶数，则将Z赋值为0，否则将Z赋值为1。
   b. 计算整数0~X的和。
5. 写一个Bare Bones语言例程，将值X除以值Y，忽略余数。也就是说，1除以2得0，5除以3得1。
6. 描述由下列Bare Bones语言程序计算的函数，假设该函数的输入由X和Y表示，输出由Z表示：

```
copy X to Z;
copy Y to Aux;
while Aux not 0 do;
  decr Z;
  decr Aux;
end;
```

7. 描述由下列Bare Bones语言程序计算的函数，假设该函数的输入由X和Y表示，输出由Z表示：

```
clear Z;
copy X to Aux1;
copy Y to Aux2;
while Aux1 not 0 do;
 while Aux2 not 0 do;
   decr Z;
   decr Aux2;
 end;
 decr Aux1;
end;
```

8. 写一个Bare Bones语言程序，用于计算变量X和变量Y的异或，并将结果存放在变量Z中。你可以假设X和Y只从整数0和1开始。
9. 如果我们允许一个Bare Bones程序中的指令可以用整数值标号，并且把while循环结构用形如

```
if name not 0 goto label;
```

的条件转移指令代替，其中，*name*是任意一个变量，label是一个整数值，用来标记别处的一条指令，那么请证明：这个新语言仍然是一种通用程序设计语言。

10. 在本章中，我们已经看到，语句

```
copy name1 to name2;
```

如何用Bare Bones语言来模拟。请证明：如果Bare Bones语言中的while循环结构用一个形如

```
repeat...until (name equals 0 )
```

的后测试循环结构来代替，那么上述语句仍能够被模拟。

11. 证明：如果while语句用一个形如

```
repeat...until (name equals 0 )
```

的后测试循环结构来代替，Bare Bones语言仍为通用程序设计语言。

12. 设计一个图灵机，要求它一旦启动，将不会使用磁带上的一个以上的单元，而且永不会到达停止状态。
13. 设计一个图灵机，要求将当前单元左边的所有单元置0，直到遇到一个包含星号的单元为止。
14. 假设图灵机的磁带上0、1模式串的两端是用星号作为分界符的，那么请设计一个图灵机，将此模式左移一个单元，这里假设机器是从模式右端的星号处开始。
15. 设计一个图灵机，要求把在当前单元（它包含有一个星号）和其左边的第一个星号之间所发现的0、1模式串倒转。
16. 概述丘奇-图灵论题。
17. 下面的Bare Bones语言程序是自终止的吗？请说明理由。

```
copy X to Y;
incr Y;
```

```
incr Y;
while X not 0 do;
   decr X;
   decr X;
   decr Y;
   decr Y;
end;
decr Y;
while Y not 0 do;
   incr X;
   decr Y;
end;
while X not 0 do;
end;
```

18. 下面的Bare Bones语言程序是自终止的吗？请说明理由。

```
while X not 0 do;
end;
```

19. 下面的Bare Bones语言程序是自终止的吗？请说明理由。

```
while X not 0 do;
  decr X;
end;
```

20. 分析下面一对命题的有效性：

```
The next statement is true.
The previous statement is false.
```

21. 分析命题“船上的厨师为所有人做饭，但只为那些自己不做饭的人做饭。”的有效性。（提示：谁为厨师做饭？）
22. 假设你在一个国家，这个国家的人要么是说真话者，要么是说假话者。（说真话的人一直说真话，说假话的人一直说假话。）那么，你怎样向一个人只问一个问题，就判断这个人是说真话者，还是说假话者。
23. 请概括说明，图灵机在理论计算机科学领域里的重要性。
24. 请概括说明，停机问题在理论计算机科学领域里的重要性。
25. 假设你要在一群人中找出是否有人在某一天过生日。一种方法就是询问这群人中的每个成员一次。如果你采用这种方法，那么出现何种情况会让你知道：存在这样一个人在那天过生日？出现何种情况会让你知道：不存在这样的人在那天过生日？现在，假设要找出至少一个具有某种特征的正整数，你可以应用同样的方法对整数一次一个地进行系统测试。事实上，如果某个整数具有这种特征，你会怎样将它找出来？然而，如果没有整数具有这样的特征，那你是怎样发现的？为了确定一个推测是否为真，是不是必须得与确定这个推测是否为假对称地进行测试？
26. 在表中查找某个值的问题属于多项式问题吗？请证明你的答案。
27. 设计一个算法，来确定给定的一个正整数是否为素数。你的解是否高效？你的解是多项式的还是非多项式的？
28. 一个问题的多项式解是不是一直都比其指数解要好？请说明理由。
29. 一个问题有多项式解这样一个事实，是否就意味着它能一直在实际可行的时间内求解？请说明理由。
30. 程序员查理要解决这样一个问题：将一个组（人数为偶数）分成人数相同的两个小组，要使得每个小组的总年龄间的差别尽可能大。他提出的解决方案是：先构建出所有可能的小组对，计算每个对总年龄间的差别，然后选取差别最大的那一对。但是，程序员玛丽提出的解决方案是：首先将初始组按年龄进行排序，再分成两个小组，年龄较小的一半为一组，年龄较大的一半为另一组。每种解决方案的复杂性是什么？这个问题本身是属于多项式复杂性、NP复杂性还是非多项式复杂性？
31. 对于表的排序问题而言，可以先生成出表的所有排列，然后再选出所需要的那一种排列。那么请问，为什么这种方法不是一个令人满意的方法？
32. 假设一种彩票基于的是正确选择4个整数值，每个值的范围都是1~50。并假设累计奖金已经大到对每种可能的组合分别买一张彩票都能获利的地步。如果买一张彩票要花一秒钟的时间，那么为每种可能的组合都买一张彩票得花多长时间？如果彩票需要选取5个数，而不是4个数，那么所需时间将如何变化？依据本章所讨论的内容，这个问题必须要做些什么？
33. 下面的算法是确定性的吗？请说明理由。

procedure mystery (Number)
if (Number > 5)
　　then (回答"yes")

else (挑一个比5小的数并将这个数作为答案)

34. 下面的算法是确定性的吗？请说明理由。

    一直向前开。
    在第三个十字路口问站。
    在拐角处的人问他是应该往右还是往左。
    根据那个人指的方向转弯。
    开两个区，然后停下来。

35. 请确定下列算法中的非确定的地方：

```
选1~100的3个数。
if (如果所选数字之和大于150)
   then (回答"yes")
   else (选择已选出的数中的一个，将该数
         作为答案)
```

36. 下列算法是具有多项式时间复杂性还是具有非多项式时间复杂性？请说明理由。

```
procedure mystery (ListOfNumbers)
从ListOfNumbers中选一组数
if (这些数加起来得125)
  then (回答"yes")
  else (不给出答案)
```

37. 以下问题中，哪些问题是属于P类的？
    a. 复杂性为n^2的问题
    b. 复杂性为$3n$的问题
    c. 复杂性为n^2+2n的问题
    d. 复杂性为$n!$的问题
38. 概述声明一个问题是多项式问题和声明一个问题是非确定性多项式问题之间的区别。
39. 请举出一个问题的例子，要求该问题既属于P类，又属于NP类。
40. 假设给你两个算法用来解决同一个问题。一个算法的时间复杂性为n^4，而另一个算法的时间复杂性为$4n$，那么在什么样规模的输入上前者比后者更为有效？
41. 假设要解决的问题是旅行商问题，其中所涉及的城市数为15，而任意两个城市之间只有一条路相连。那么请问，遍历这些城市共有多少种不同的路径？这里假设每条路径的长度能够在一微秒内计算出来，那么计算出所有这些路径的长度要花多长时间？
42. 如果将归并排序算法（如图12-15和图12-4所示）应用到表Alice、Bob、Carol以及David，那么要做多少次名字比较？如果应用到表Alice、Bob、Carol、David以及Elaine，那么又将做多少次名字比较？
43. 请为图12-12所示的每一类问题都举出一个例子。
44. 设计一个算法，找到形如$x^2 + y^2 = n$的等式的整数解，这里，n是某个给定的正整数。并确定你的算法的时间复杂性。
45. **背包问题**（knapsack problem）是另一个属于NP完全类的问题。该问题旨在从一个表中找出一些数，使得这些数的和等于某个值。例如，表

```
642  257  771  388  391  782  304
```

中的数据项257、388和782，它们的和为1 427。请找出和为1 723的数据项。你用的是什么算法？其复杂性又如何？

46. 请指出旅行商问题与背包问题的相似之处（参见习题45）。
47. 下面的表排序算法称为冒泡排序。当将其用到一个有n个数据项的表中时，冒泡排序需要在表项中进行多少次比较？

```
procedure BubbleSort(List)
Counter ← 1;
while (Counter < List中的项的个数) do
  [N ← List中的项的个数;
   while (N > 1) do
     (if(List中的第N个项小于其前面的项);
      then(第N个项和前面的一项交换)
      N ← N-1
      )
  ]
```

48. 用RSA公钥加密技术对报文110进行加密，这里公钥为n=91和e=5。
49. 用RSA公钥加密技术对报文111进行解密，这里私钥为n=133和d=5。
50. 假设你知道一个基于RSA算法的公钥加密系统，其公钥为n = 77和e =7。私钥是什么？怎样才能让你在一个合理的时间范围内解决这个问题？
51. 找出107 531的因数。这个问题与本章的内容有什么关联？
52. 如果正整数n在2~$\sqrt{n}$范围内没有整数因子，那么我们能够得出什么结论？对于找一个正整数的因子这样的任务来说，这又能告诉你一些什么？

社会问题

下面的问题有助于你分析一些与计算领域相关的道德、社会和法律问题。回答这些问题不是唯一的目的，还应该考虑为什么这样回答，以及你的判断是否对每个问题都标准如一。

1. 假设一个问题的最优算法解将需要执行100年，那么你会认为这个问题是容易处理的吗？为什么？
2. 公民有权在免受政府部门监控的情况下对报文进行加密吗？你的回答是否提供了“合适的”法律执行？那么谁来决定“合适的”法律执行是什么？
3. 如果人脑是一个算法设备，那么关于人性，图灵论题会有什么结果？在何种程度上你会认为图灵机包含了人脑的计算能力？
4. 我们已经看到，不同的计算模型（如有限表、代数公式、图灵机等）有不同的计算能力。不同的生物体也有不同的计算能力吗？不同的人的计算能力也不同吗？如果是这样，那么具有较高能力的人是否能用这些能力来获得更好的生活方式？
5. 在今天，有许多网站提供了大多数城市的交通地图。这些站点能够帮助寻找某个具体的地址，并提供放大功能来看清小范围的街区布局。从这个事实出发，考虑下面一系列的假想。假设这些地图站点配备了具有变焦功能的卫星拍照技术。假设这些变焦功能增强到能对某个独立的建筑及其周边的景象给出更为详细的图像。假设这些图像又增强到包括实时视频。假设这些实时视频图像通过使用红外线技术而得到加强。到了这个地步，别人就能够一天24小时地监视你的家。在这一系列技术进步中，你的隐私权是在哪一点开始受到侵犯的？在这一系列技术进步中，你认为什么事情上我们的做法超越了当今间谍卫星技术的能力？在怎样的程度上，这个场景就只是假想的？
6. 假设一个公司开发了一个加密系统，并取得了专利权。政府机构是否有权以国家安全的名义来使用这个系统？政府机构是否有权以国家安全的名义限制这个公司对这个系统的商业使用？如果这个公司是跨国公司，那么情况又会怎样？
7. 假设你买了一个产品，其内部结构是加密的，那么你是否有权对这个产品的基本结构进行解密？如果是，你是否有权以商业方式来使用这些信息？以非商业方式使用呢？如果它的加密是利用一个秘密的加密系统来实现的，而你发现了这个秘密，那么你有权分享这个秘密吗？
8. 很多年以前，哲学家约翰·杜威（1859—1952）提出了“履责技术”（responsible technology）这个术语。请举出一些例子，来说明你对“履责技术”是怎样理解的。在例子的基础上，阐明你自己对“履责技术”的定义。在过去的100多年里，社会实践了“履责技术”吗？应当采取措施来保证它的实施吗？如果是，则应采取什么样的措施？如果不是，为什么？

课外阅读

Garey, M. R. and D. S. Johnson. *Computers and Intractability*. New York: W. H. Freeman, 1979.

Hamburger, H. and D. Richards. *Logic and Language Models for Computer Science*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

Hofstadter, D. R. *Gödel, Escher, Bach: An Eternal Golden Braid*. St. Paul, MN: Vintage, 1980.

Hopcroft, J. E., R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Boston, MA: Addison-Wesley, 2007.

Lewis, H. R. and C. H. Papadimitriou. *Elements of the Theory of Computation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall,1998.

Rich E. Automata. *Computability, and Complexity: Theory and Application*. Upper Saddle River, NJ: Prentice-Hall, 2008.

Sipser, M. *Introduction to the Theory of Computation*. Boston: PWS, 1996.

Smith, C. and E. Kinber. *Theory of Computing: A Gentle Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 2001.

Sudkamp , T. A. *Languages and Machines: An Introduction to the Theory of Computer Science*, 3rd ed. Boston, MA: Addison-Wesley, 2006.

附录A

ASCII码

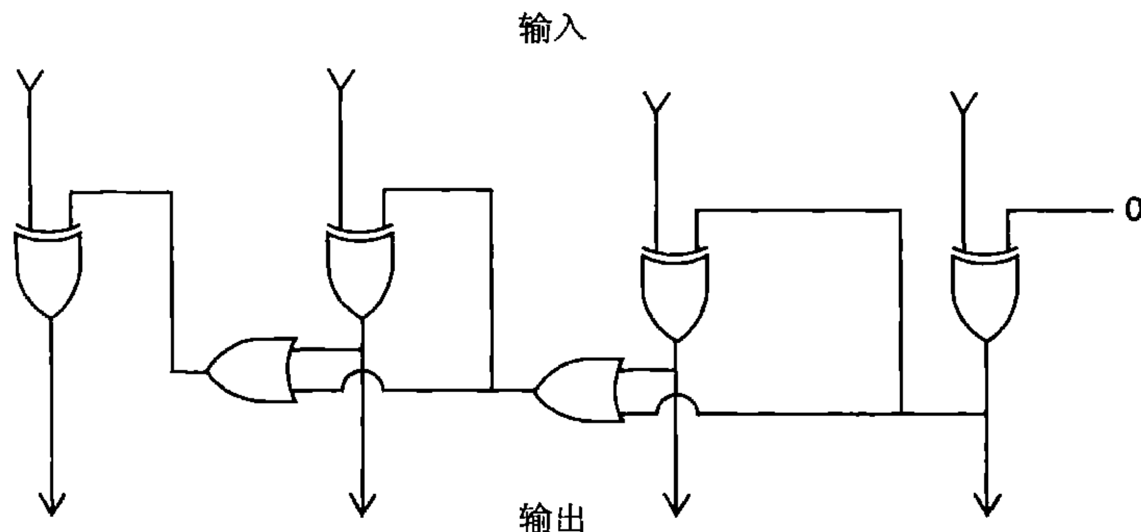
下面列出了ASCII码的一部分，并在每个位模式的左边都添加了一个0，以构成现在通用的8位位模式。第3列是每个8位位模式对应的十六进制值。

符号	ASCII	十六进制值	符号	ASCII	十六进制值	符号	ASCII	十六进制值
空行	00001010	0A	>	00111110	3E	^	01011110	5E
回车	00001011	0B	?	00111111	3F	_	01011111	5F
空格	00100000	20	@	01000000	40	`	01100000	60
!	00100001	21	A	01000001	41	a	01100001	61
"	00100010	22	B	01000010	42	b	01100010	62
#	00100011	23	C	01000011	43	c	01100011	63
$	00100100	24	D	01000100	44	d	01100100	64
%	00100101	25	E	01000101	45	e	01100101	65
&	00100110	26	F	01000110	46	f	01100110	66
'	00100111	27	G	01000111	47	g	01100111	67
(	00101000	28	H	01001000	48	h	01101000	68
)	00101001	29	I	01001001	49	i	01101001	69
*	00101010	2A	J	01001010	4A	j	01101010	6A
+	00101011	2B	K	01001011	4B	k	01101011	6B
'	00101100	2C	L	01001100	4C	l	01101100	6C
-	00101101	2D	M	01001101	4D	m	01101101	6D
.	00101110	2E	N	01001110	4E	n	01101110	6E
/	00111111	2F	O	01001111	4F	o	01101111	6F
0	00110000	30	P	01010000	50	p	01110000	70
1	00110001	31	Q	01010001	51	q	01110001	71
2	00110010	32	R	01010010	52	r	01110010	72
3	00110011	33	S	01010011	53	s	01110011	73
4	00110100	34	T	01010100	54	t	01110100	74
5	00110101	35	U	01010101	55	u	01110101	75
6	00110110	36	V	01010110	56	v	01110110	76
7	00110111	37	W	01010111	57	w	01110111	77
8	00111000	38	X	01011000	58	x	01111000	78
9	00111001	39	Y	01011001	59	y	01111001	79
:	00111010	3A	Z	01011010	5A	z	01111010	7A
;	00111011	3B	[	01011011	5B	{	01111011	7B
<	00111100	3C	\	01011100	5C	|	01111100	7C
=	00111101	3D	]	01011101	5D	}	01111101	7D

附录B

处理二进制补码表示的电路

本附录介绍用电路实现对二进制补码表示的值进行取负及相加操作。我们从图B-1的电路开始，它将一个4位的二进制补码转换为该值的负值的表示。例如，假设给出3的二进制补码表示，该电路则产生-3的表示。算法与正文中所述一样。也就是说，它自右向左复制位模式，直至复制到一个1，然后当从左向右移动时，求剩余各位的补码。由于最右边XOR门的一个输入值固定为0，所以此门只能将其另一个输入传送至输出。然而这个输出又向左传给下一个XOR门作为一个输入。如果这个输出是1，那么下一个XOR门将会把其输入位取反后送给输出。而且，这个1还会通过OR门向左传送，影响下一个门。就这样，复制给输出的第一个1也会向左传送，使得所有剩余的位在送到输出时都取反。



图B-1　将一个二进制补码位模式取负的电路

接下来，我们考虑一下用二进制补码表示的两个数值的加法。具体而言，解决问题

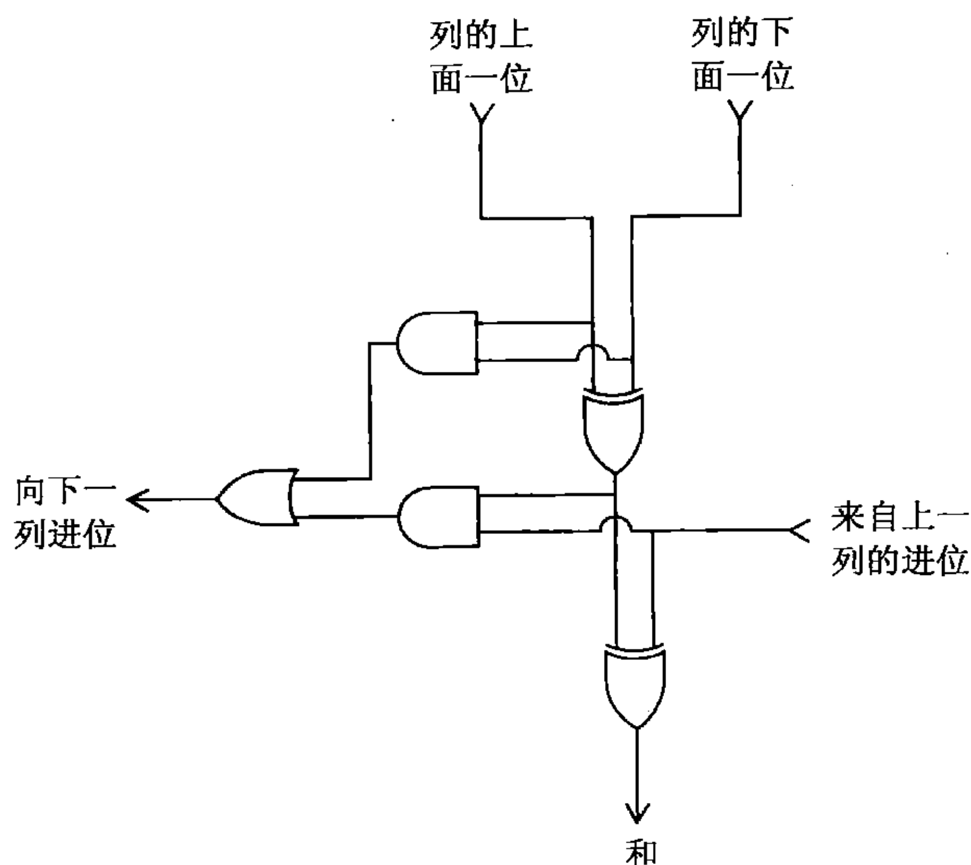
```
+ 0110
------
+ 1011
```

时，我们是从右至左一列一列地计算，并且每列都执行相同的算法。因此，只要获得这类问题一列的加法电路，通过重复这个单列电路，就可以构建许多列的相加电路。

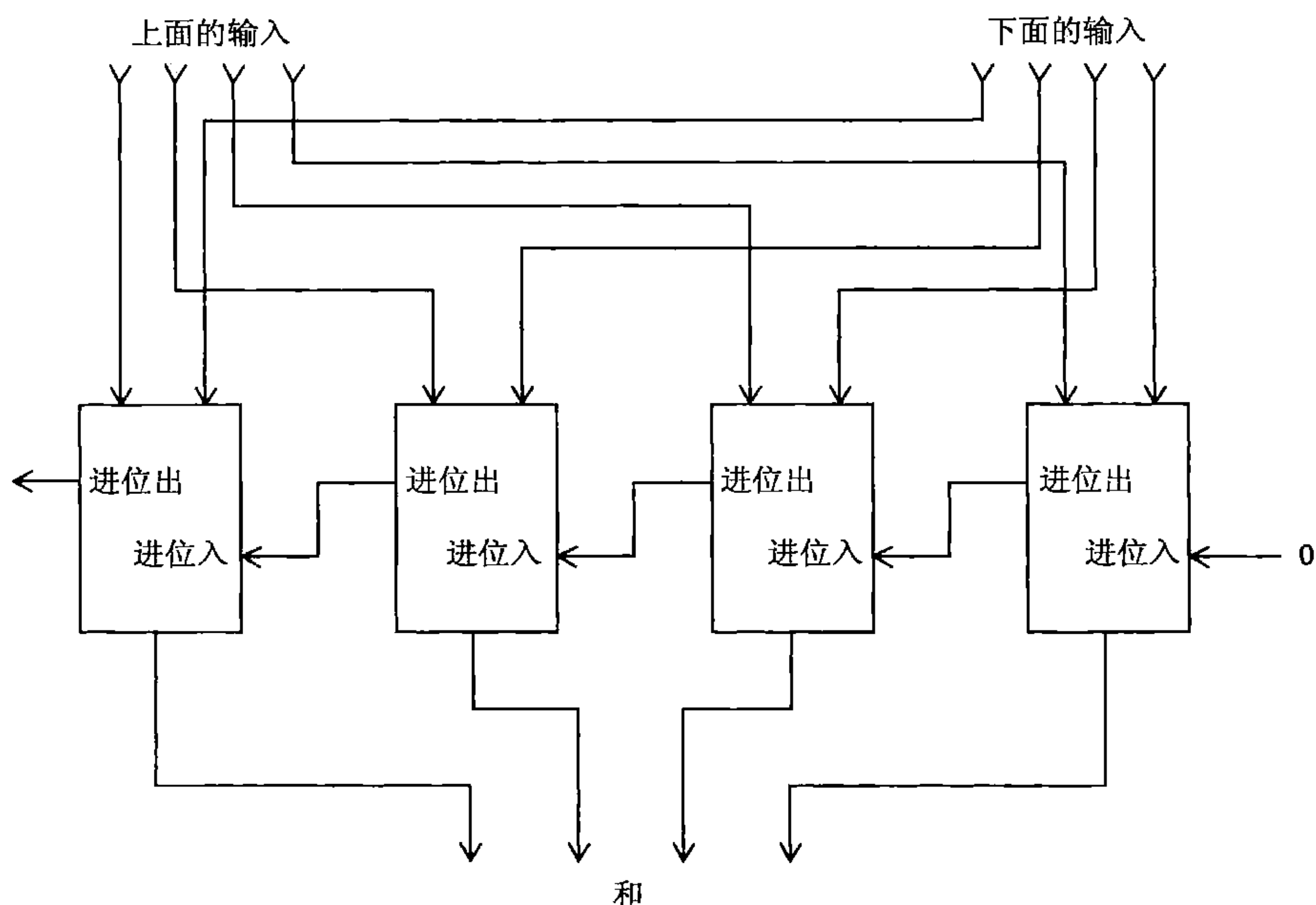
多列加法问题中一个单列的相加算法是这样的：把当前列的两个数值相加，把相加的和加上上一列的进位，并将此和的最低有效位记为答案，然后将进位传向下一列。图B-2中的电路遵循这个算法。上面的XOR门决定了两个输入位的和。下面的XOR门将这个和与上一列的进位相加。两个AND门及OR门一起把进位向左传送。具体来说，如果该列中最初的两个输入是1，或者这些位的和及进位都是1，那么就会产生一个进位1。

图B-3表示单列电路的副本如何用于产生计算用4位二进制补码系统表示的两个值和的电路。图中每个矩形表示单列加法电路的一个副本。需要注意的是：最右边矩形的进位值总是0，

因为它没有来自上一列的进位。以此类推，最左边矩形产生的进位将被忽略。



图B-2　在一个多列加法问题中进行单列相加的电路



图B-3　使用图B-2中电路的4个副本将2个二进制补码表示的数值相加的电路

因为进位的信息自最右向最左传播，或者说波动，所以图B-3电路称为行波进位加法器（ripple adder）。这类电路尽管结构简单，但执行速度较慢，而一些更好的电路，例如，先行进位加法器，就可以将这种列到列的传播减到最小。于是，尽管图B-3中的电路足够满足我们的要求，但并不是当今机器中使用的电路。

附录C

一种简单的机器语言

本附录介绍一种简单却有代表性的机器语言。我们首先解释一下这一机器本身的体系结构。

C.1 机器体系结构

这种机器有16个通用寄存器，编号为0~F（十六进制表示）。每个寄存器的长度为1字节（8位）。为了在指令中标识寄存器，每个寄存器被赋予了唯一的4位模式，用于代表其寄存器号。于是，寄存器0由0000（十六进制0）标识，寄存器4由0100（十六进制4）标识。

机器主存中有256个单元，每个单元被赋予一个范围为0~255的整数地址。因此，一个地址可以由00000000~11111111（或者是十六进制值的00~FF）的一个8位模式来表示。

假设浮点数值以1.7节讨论过并且概括在图1-26中的8位格式存储。

C.2 机器语言

每条机器指令都是2字节长：前面的4位是操作码，后面的12位组成操作数字段。下面的表格列出了用十六进制记数法表示的指令及简要说明。字母R、S及T在表示寄存器标识符的那些字段处用来替代十六进制数字，并且因指令的具体应用而异。字母X及Y用来在变量字段替代十六进制数字，而不是代表寄存器。

操作码	操作数	说明
1	RXY	在地址为XY的存储单元中找到的位模式装载（LOAD）寄存器R 例：14A3将地址为A3的存储单元的内容放入寄存器4
2	RXY	以位模式XY装载（LOAD）寄存器R 例：20A3将数值A3放入寄存器0
3	RXY	将寄存器R中的位模式存放（STORE）在地址为XY的存储单元中 例：35B1将使得寄存器5中的内容放入地址为B1的存储单元
4	0RS	将寄存器R中的位模式移入（MOVE）寄存器S 例：40A4将寄存器A的内容复制到寄存器4
5	RST	将寄存器S及寄存器T的位模式作为二进制补码表示相加（ADD），求和结果存放在寄存器R中 例：5726将寄存器2和寄存器6中的二进制数值相加，并将结果存放在寄存器7中

（续）

操作码	操作数	说明
6	RST	将寄存器S及寄存器T的位模式作为浮点表示值相加（ADD），并将浮点结果存放在寄存器R中 例：634E将寄存器4和寄存器E中的浮点值相加，并将结果存放在寄存器3中
7	RST	将寄存器S及寄存器T中的位模式做或（OR）操作，并将结果存放在寄存器R中 例：7CB4将寄存器B和寄存器4的内容做或操作，结果存放在寄存器C中
8	RST	将寄存器S及寄存器T中的位模式做与（AND）操作，并将结果存放在寄存器R中 例：8045将寄存器4和寄存器5的内容做与操作，结果存放在寄存器0中
9	RST	将寄存器S和寄存器T中的位模式进行异或（EXCLUSIVE OR）操作，并将结果存放在寄存器R中 例：95F3将使得寄存器F和寄存器3的内容进行异或操作，结果存放在在寄存器5中
A	ROX	将寄存器R中的位模式循环（ROTATE）右移一位，进行X次。每次都把从低位端开始的那个位放入高端 例：A403将使得寄存器4中的内容循环右移3位
B	RXY	如果寄存器R中的位模式等于寄存器0中的位模式，那么转移（JUMP）到地址XY处的存储单元中的指令。否则，继续正常的执行顺序（转移是通过在执行周期将XY复制到程序计数器来实现的） 例：B43C将首先比较寄存器4和寄存器0中的内容。如果二者相等，则把模式3C放入程序计数器，所以下一条执行的指令将是这个存储地址中的那条。否则，不做任何事情，程序将照常继续
C	000	停止（HALT）执行 例：C000将使得程序停止执行

附录D

高级编程语言

本附录包含了在第6章中作为例子使用的每种语言的简要背景。

D.1 Ada 语言

Ada语言是根据奥古斯塔·艾达·拜伦（Augusta Ada Byron）（1815—1851）命名的。她是查尔斯·巴贝奇（Charles Babbage）的拥护者，诗人拜伦勋爵（Lord Byron）的女儿。这个语言最初是由美国国防部开发的，目的是为了得到一种满足其所有软件开发需要的通用语言。在Ada的设计期间，一个重点是加入实时计算机系统编程的特性，这类系统常用来作为更大型机器的一部分，如导弹遥控系统、楼际间的环境控制系统以及汽车和小型家用电器中的控制系统。因此，Ada语言包含这样一些特征：既可以表达并行处理环境中的活动，又可以作为合适的技术解决在应用环境中出现的特殊情况（称为异常）。尽管初期是作为命令式语言设计的，但Ada的新版本中包含了面向对象范型。

Ada语言的设计一贯强调可实现可靠软件高效开发的特性，下面这个事实例证了这个特性：波音777飞机中所有内部的控制软件都是用Ada语言编写的，这也是Ada用作SPARK语言开发（正如第5章中说明的）的起始点的主要原因。

D.2 C 语言

C语言在20世纪70年代初期由贝尔实验室的Dennis Ritchie开发。尽管它最初是作为开发系统软件的语言来设计的，但却在程序设计界颇为流行，并且已经被美国国家标准化组织标准化。

C语言最初的设想只是跨出机器语言的一步而已。所以与使用完整的英语词汇的其他高级语言相比，它的语法十分简洁，原语都用专门符号表示。它的这种简练性可有效地表示复杂算法，这也是其广为流行的一个主要原因。（通常说来，简洁的表示比冗长的表达更易读。）

D.3 C++

C++由贝尔实验室的Bjarne Stroustrup开发，它是作为C语言的一个增强版本开发的。目的是开发一种可以与面向对象范型相兼容的语言。当今，C++凭其自身的实力已经成为著名的面向对象语言，同时它还作为另外两种主流面向对象语言（Java和C#）开发的起始点。

D.4 C#

C#是由微软公司开发的.NET框架中的工具。这是为运行微软系统软件的机器开发应用软件

的一个综合系统。C#看起来很像C++或Java程序。事实上，微软公司将C#作为一种不同的语言来推介并不是因为它是一种全新的语言，而是因为它可以根据自己的需要定制语言的一些专门特性，而不必考虑与其他语言相关的标准，也不需要考虑其他公司的专利权问题。因此，C#的创新性在于：在利用.NET框架开发软件中，它的作用非常突出。拥有微软公司的支持，在未来的几年里，C#以及.NET框架一定能够成为软件开发界的佼佼者。

D.5 Fortran

FORTRAN是FORmula TRANslator（公式翻译语言）的缩写。该语言是第一批开发的高级语言之一（公布于1957年），并且是在计算界中最先获得广泛认同的语言之一。多年以来，它的官方描述经历了许多次的扩充，也就是说，今天的FORTRAN语言与原始的版本有很大的不同。事实上，通过学习FORTRAN语言的演变，人们能见证研究对程序语言设计产生的影响。尽管最初是设计成一种指令语言，但FORTRAN的新版本现在已经包含了许多面向对象特性。FORTRAN语言在科学界仍然是一种很流行的语言。具体来说，许多数值分析以及统计软件包都是使用FORTRAN语言编写的，而且仍将继续用FORTRAN语言编写。

D.6 Java

Java是Sun公司在20世纪90年代早期开发的一种面向对象语言。该语言的设计者在很大程度上借了C语言以及C++语言的特性。Java带来的振奋不是由于语言本身，而是由于该语言的通用实现性以及在Java编程环境中大量预先设计好的模板。通用实现性指的是用Java语言编写的程序能够在很多机器上有效地执行；模板的可用性意味着复杂软件能相对比较容易开发出来。例如，applet（小应用程序）及servlet（小服务程序）等模板使得万维网软件的开发更加流畅。

附录E

迭代结构与递归结构的等价性

在本附录中，我们使用第11章中的Bare Bones语言作为工具来回答第4章中提出的关于迭代结构与递归结构孰更强大的问题。回想一下：Bare Bones语言只包含3个赋值语句（clear、incr以及decr）和一个控制结构（由while-end语句对构成）。并且这种简单的语言与图灵机具有相同的计算能力；因此，如果我们接受了丘奇-图灵论题，就可以得出结论：任何具有算法解的问题都有可用Bare Bones表达的解。

迭代结构与递归结构进行比较的第一步是将Bare Bones语言的迭代结构替换成递归结构。方法如下：将while和end语句从该语言中移出，并在它们的位置提供可以把Bare Bones程序分成部分单元的能力，以及可从程序其他地方调用这些单元之一的能力。严格地说，我们建议用修改后的语言编写的每个程序由许多语法上分离的程序单元构成。假定每个程序必须正好包含严格称为MAIN的单元，它的语法结构如下：

```
MAIN: begin;
        .
        .
        .
      end;
```

（其中点表示其他Bare Bones语句）也可能包括具有以下结构的其他单元（语法上从属于MAIN），：

```
unit:  begin;
         .
         .
         .
       return;
```

（*unit*表示单元的名称，与变量名具有相同的语法。）这种分割结构的语义是：程序总是从MAIN单元的开头开始执行，并且在到达该单元的end语句时停止。除了MAIN之外的程序单元都可以通过条件语句

```
if name not 0 perform unit;
```

作为过程来调用（其中*name*表示任何变量名，*unit*表示除了MAIN之外的其他任何程序单元名）。而且，还允许MAIN单元之外的其他单元递归地调用自己。

有了这些附加的特性，我们就可以模拟原来Bare Bones中的while-end结构了。例如，一个如下形式的Bare Bones程序：

```
while X not 0 do;
S;
```

```
    end;
```

（其中*S*表示任何Bare Bones语句的序列），它可以被如下单元结构替代：

```
MAIN:  begin;
         if X not 0 perform unitA;
       end;
unitA: begin;
         S;
         if X not 0 perform unitA;
       return;
```

因此，我们得出结论：修改后的语言具有原始Bare Bones语言的所有能力。

也可以说明，任何使用修改后语言解决的问题都（能够）用Bare Bones来解决。做到这一点的一种方法就是说明任何用修改后语言表达的算法都可以用原始的Bare Bones语言编写。不过，这涉及递归结构如何用Bare Bones语言的while-end结构来模拟的一个比较精确的描述。

对于我们的目的，比较简单的就是根据第11章所介绍的丘奇-图灵论题。具体来说，丘奇-图灵论题，加上Bare Bones与图灵机器具有相同的能力这个事实，表明了没有比原始Bare Bones更强大的语言了。所以，任何可以用我们修改后的语言求解的问题也能用Bare Bones解决。

我们得出的结论是，修改后语言的能力与原始Bare Bones的能力一样。两种语言之间的唯一区别是：一种提供的是迭代控制结构，而另外一种提供的是递归控制结构。因此，实际上这两种控制结构在计算能力上是等价的。

索　引

符号

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

W

X

Y

Z